

DR



Candy



Candy



Candy



Table



Staircase



Robot



Nuts



Nuts



Coffee



Crackers



Mustard



Cup



Robot





# Project 0

---

- Instructions and code available on the website
  - Here: [deeprob.org/projects/project0/](https://deeprob.org/projects/project0/)
- Uses Python, PyTorch and Google Colab
- Introduction to PyTorch Tensors
- Due this Thursday (January 12th), 11:59 PM EST





# Project 0 Suggestions

---

- If you choose to develop locally
  - **PyTorch Version 1.13.0**
- Ensure you save your notebook file before uploading submission
- Close any Colab notebooks not in use to avoid usage limits





# Discussion Forum

---

- Ed Stem available for course discussion and questions
  - Forum is shared across UMich and UMinn students
  - Participation and use is not required
  - Opt-in using [this Google form](#)
  - **Discussion of quizzes and verbatim code must be private**





# Enrollment

---

- Additional class permissions being issued
- Both sections (498 & 599)
- If you haven't received a class permission contact Anthony and Prof. Jenkins





# Image Classification



# Image Classification—A Core Computer Vision Task

**Input:** image



**Output:** assign image to one of a fixed set of categories

**Chocolate Pretzels**

Granola Bar

Potato Chips

Water Bottle

Popcorn

# Problem—Semantic Gap

**Input:** image



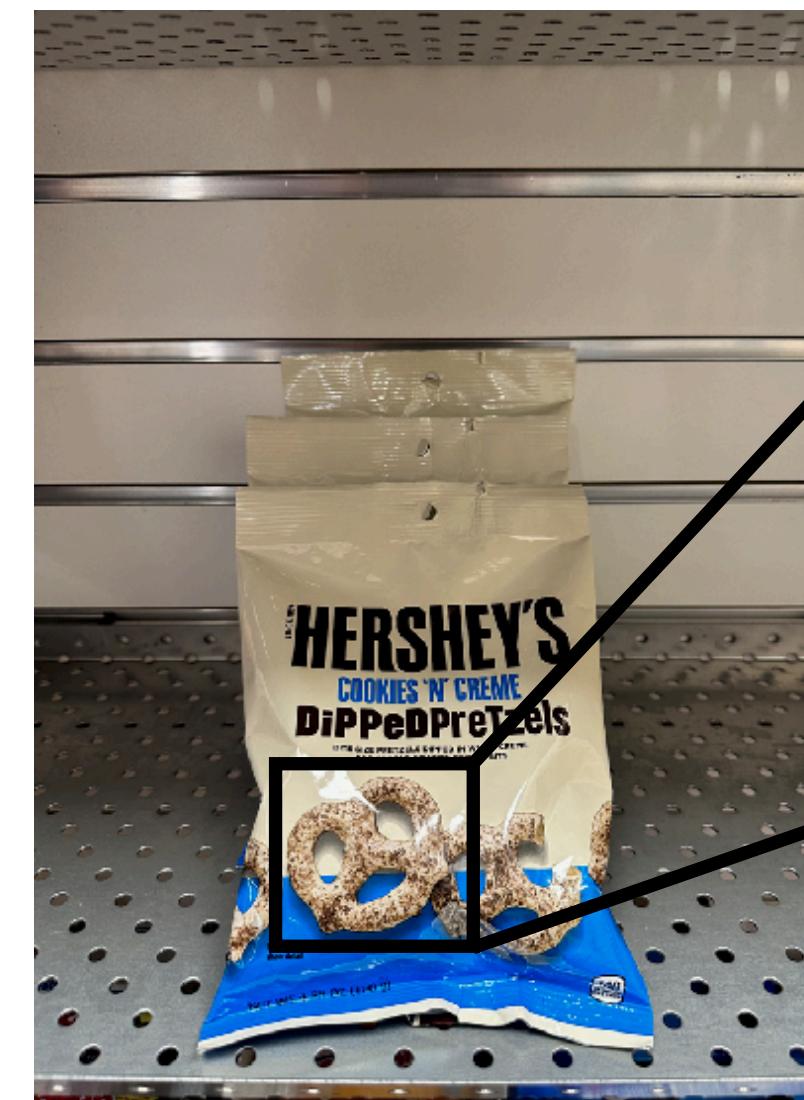
```
[183, 187, 189, 189, 188, 188, 189, 190, 186, 185, 189, 190, 187, 186, 183],  
[185, 188, 189, 188, 188, 189, 191, 193, 187, 190, 191, 189, 186, 185, 185],  
[186, 189, 189, 187, 187, 188, 189, 189, 192, 194, 189, 184, 182, 185, 187],  
[188, 188, 188, 190, 190, 189, 189, 190, 190, 189, 185, 184, 185, 188, 188],  
[187, 187, 188, 192, 191, 189, 191, 193, 191, 186, 185, 189, 187, 187, 185],  
[186, 186, 189, 191, 190, 189, 190, 192, 191, 188, 190, 193, 186, 186, 184],  
[189, 186, 189, 192, 192, 190, 191, 193, 184, 188, 190, 192, 186, 187, 186],  
[191, 189, 189, 190, 189, 190, 190, 183, 187, 186, 188, 187, 189, 188],  
[192, 194, 193, 189, 188, 193, 194, 191, 191, 192, 186, 186, 187, 186, 187],  
[190, 192, 193, 191, 191, 195, 194, 191, 191, 192, 188, 189, 189, 186, 188],  
[189, 188, 190, 189, 190, 189, 187, 187, 185, 190, 188, 189, 192, 192, 191],  
[191, 188, 187, 186, 188, 190, 189, 190, 186, 193, 190, 187, 194, 194, 192],  
[194, 193, 189, 186, 189, 190, 191, 194, 192, 191, 192, 194, 194, 194, 188],  
[196, 196, 196, 193, 191, 190, 191, 195, 194, 191, 193, 194, 192, 190, 187],  
[194, 193, 194, 191, 188, 189, 190, 193, 193, 191, 193, 192, 190, 190, 190],  
[197, 194, 193, 191, 188, 189, 191, 192, 192, 194, 192, 190, 193, 193],  
[202, 201, 202, 200, 196, 193, 192, 192, 190, 191, 194, 193, 191, 193, 193],  
[205, 206, 207, 206, 202, 198, 196, 194, 189, 190, 191, 192, 191, 191, 190],  
[207, 207, 204, 202, 199, 198, 199, 199, 195, 192, 192, 194, 193, 191, 190],  
[205, 203, 200, 200, 199, 196, 198, 202, 199, 194, 193, 195, 193, 191, 192],  
[199, 196, 196, 201, 205, 204, 202, 199, 194, 192, 193, 191, 189, 192],  
[195, 194, 193, 196, 201, 205, 205, 203, 200, 196, 195, 195, 192, 190, 192],  
[194, 194, 193, 194, 196, 199, 202, 204, 201, 200, 200, 199, 196, 195, 196],  
[194, 193, 192, 195, 197, 199, 202, 204, 200, 203, 204, 202, 199, 200, 200],  
[199, 201, 201, 200, 200, 201, 201, 205, 202, 206, 207, 205, 203, 205, 203]]
```

What the computer sees

An image is just a grid of numbers between [0, 255]

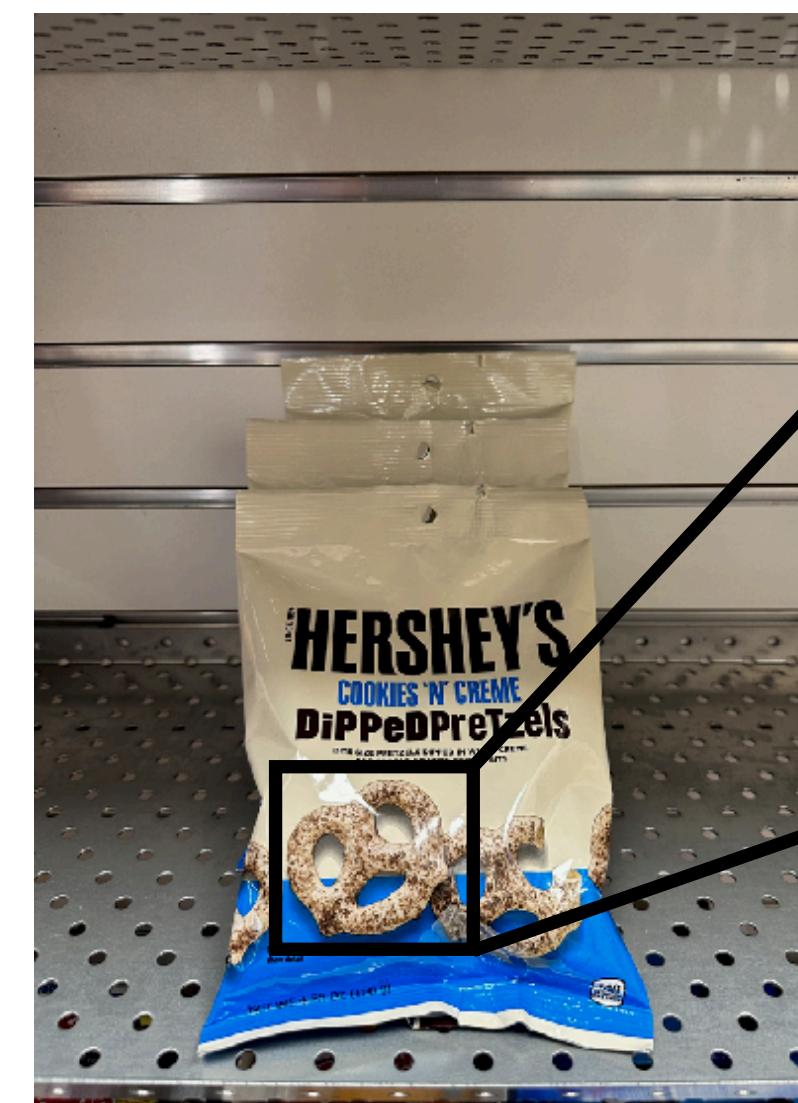
e.g. 800 x 600 x 3  
(3 channels RGB)

# Challenges—Viewpoint Variation



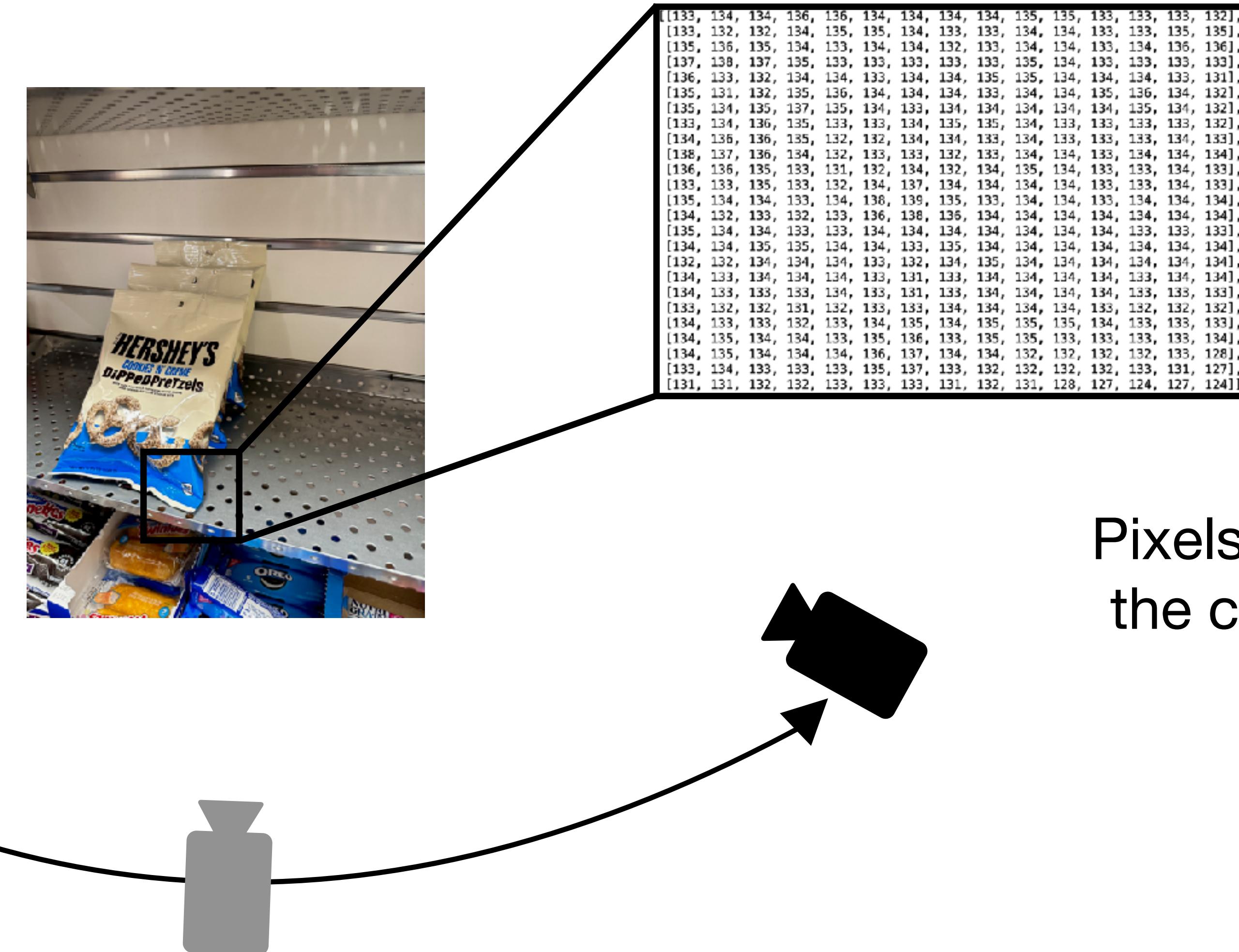
```
[183, 187, 189, 189, 188, 188, 189, 190, 186, 185, 189, 190, 187, 186, 183],  
[185, 188, 189, 188, 188, 189, 191, 193, 187, 190, 191, 189, 186, 185, 185],  
[186, 189, 189, 187, 187, 188, 189, 192, 194, 189, 184, 182, 185, 187],  
[188, 188, 188, 190, 190, 189, 190, 190, 189, 185, 184, 185, 188, 188],  
[187, 187, 188, 192, 191, 189, 191, 193, 191, 186, 185, 189, 187, 187, 185],  
[186, 186, 189, 191, 190, 189, 190, 192, 191, 188, 190, 193, 186, 186, 184],  
[189, 186, 189, 192, 192, 190, 191, 193, 184, 188, 190, 192, 186, 187, 186],  
[191, 189, 189, 190, 189, 190, 190, 183, 187, 186, 188, 187, 189, 188],  
[192, 194, 193, 189, 188, 193, 194, 191, 191, 192, 186, 186, 187, 186, 187],  
[190, 192, 193, 191, 191, 195, 194, 191, 191, 192, 188, 189, 189, 186, 188],  
[189, 188, 190, 189, 190, 189, 187, 187, 185, 190, 188, 189, 192, 192, 191],  
[191, 188, 187, 186, 188, 190, 189, 190, 186, 193, 190, 187, 194, 194, 192],  
[194, 193, 189, 186, 189, 190, 191, 194, 192, 191, 192, 194, 194, 194, 188],  
[196, 196, 196, 193, 191, 190, 191, 195, 194, 191, 193, 194, 192, 190, 187],  
[194, 193, 194, 191, 188, 189, 190, 193, 193, 191, 193, 192, 190, 190, 190],  
[197, 194, 193, 191, 188, 189, 191, 192, 192, 194, 192, 190, 193, 193],  
[202, 201, 202, 200, 196, 193, 192, 192, 190, 191, 194, 193, 191, 193, 193],  
[205, 206, 207, 206, 202, 198, 196, 194, 189, 190, 191, 192, 191, 191, 190],  
[207, 207, 204, 202, 199, 198, 199, 199, 195, 192, 192, 194, 193, 191, 190],  
[205, 203, 200, 200, 199, 196, 198, 202, 199, 194, 193, 195, 193, 191, 192],  
[199, 196, 196, 201, 205, 204, 202, 202, 199, 194, 192, 193, 191, 189, 192],  
[195, 194, 193, 196, 201, 205, 205, 203, 200, 196, 195, 195, 192, 190, 192],  
[194, 194, 193, 194, 196, 199, 202, 204, 201, 200, 200, 199, 196, 195, 196],  
[194, 193, 192, 195, 197, 199, 202, 204, 200, 203, 204, 202, 199, 200, 200],  
[199, 201, 201, 200, 200, 201, 201, 205, 202, 206, 207, 205, 203, 205, 203]
```

# Challenges—Viewpoint Variation



```
[183, 187, 189, 189, 188, 188, 189, 190, 186, 185, 189, 190, 187, 186, 183],  
[185, 188, 189, 188, 188, 189, 191, 193, 187, 190, 191, 189, 186, 185, 185],  
[186, 189, 187, 187, 188, 189, 192, 194, 184, 182, 185, 187],  
[188, 188, 188, 190, 190, 189, 190, 190, 189, 185, 184, 185, 188],  
[187, 187, 188, 192, 191, 189, 191, 193, 191, 186, 185, 189, 187, 187, 185],  
[186, 186, 189, 191, 190, 189, 190, 192, 191, 188, 190, 193, 186, 186, 184],  
[189, 186, 189, 192, 192, 190, 191, 193, 184, 188, 190, 192, 186, 187, 186],  
[191, 189, 189, 190, 189, 190, 190, 183, 187, 186, 188, 187, 189, 188],  
[192, 194, 193, 189, 188, 193, 194, 191, 191, 192, 186, 186, 187, 186, 187],  
[190, 192, 193, 191, 191, 195, 194, 191, 191, 192, 188, 189, 189, 186, 188],  
[189, 188, 190, 189, 190, 189, 187, 187, 185, 190, 188, 189, 192, 192, 191],  
[191, 188, 187, 186, 188, 190, 189, 190, 186, 193, 190, 187, 194, 194, 192],  
[194, 193, 189, 186, 189, 190, 191, 194, 192, 191, 192, 194, 194, 194, 188],  
[196, 196, 196, 193, 191, 190, 191, 195, 194, 191, 193, 194, 192, 190, 187],  
[194, 193, 194, 191, 188, 189, 190, 193, 193, 191, 193, 192, 190, 190, 190],  
[197, 194, 193, 191, 188, 189, 191, 192, 192, 194, 192, 190, 193, 193],  
[202, 201, 202, 200, 196, 193, 192, 192, 190, 191, 194, 193, 191, 193, 193],  
[205, 206, 207, 206, 202, 198, 196, 194, 189, 190, 191, 192, 191, 191, 190],  
[207, 207, 204, 202, 199, 198, 199, 199, 195, 192, 192, 194, 193, 191, 190],  
[205, 203, 200, 200, 199, 196, 198, 202, 199, 194, 193, 195, 193, 191, 192],  
[199, 196, 196, 201, 205, 204, 202, 202, 199, 194, 192, 193, 191, 189, 192],  
[195, 194, 193, 196, 201, 205, 205, 203, 200, 196, 195, 195, 192, 190, 192],  
[194, 194, 193, 194, 196, 199, 202, 204, 201, 200, 200, 199, 196, 195, 196],  
[194, 193, 192, 195, 197, 199, 202, 204, 200, 203, 204, 202, 199, 200, 200],  
[199, 201, 201, 200, 200, 201, 201, 205, 202, 206, 207, 205, 203, 205, 203]
```

# Challenges – Viewpoint Variation



# Challenges—Intraclass Variation



# Challenges—Fine-Grained Categories

Milk  
Chocolate



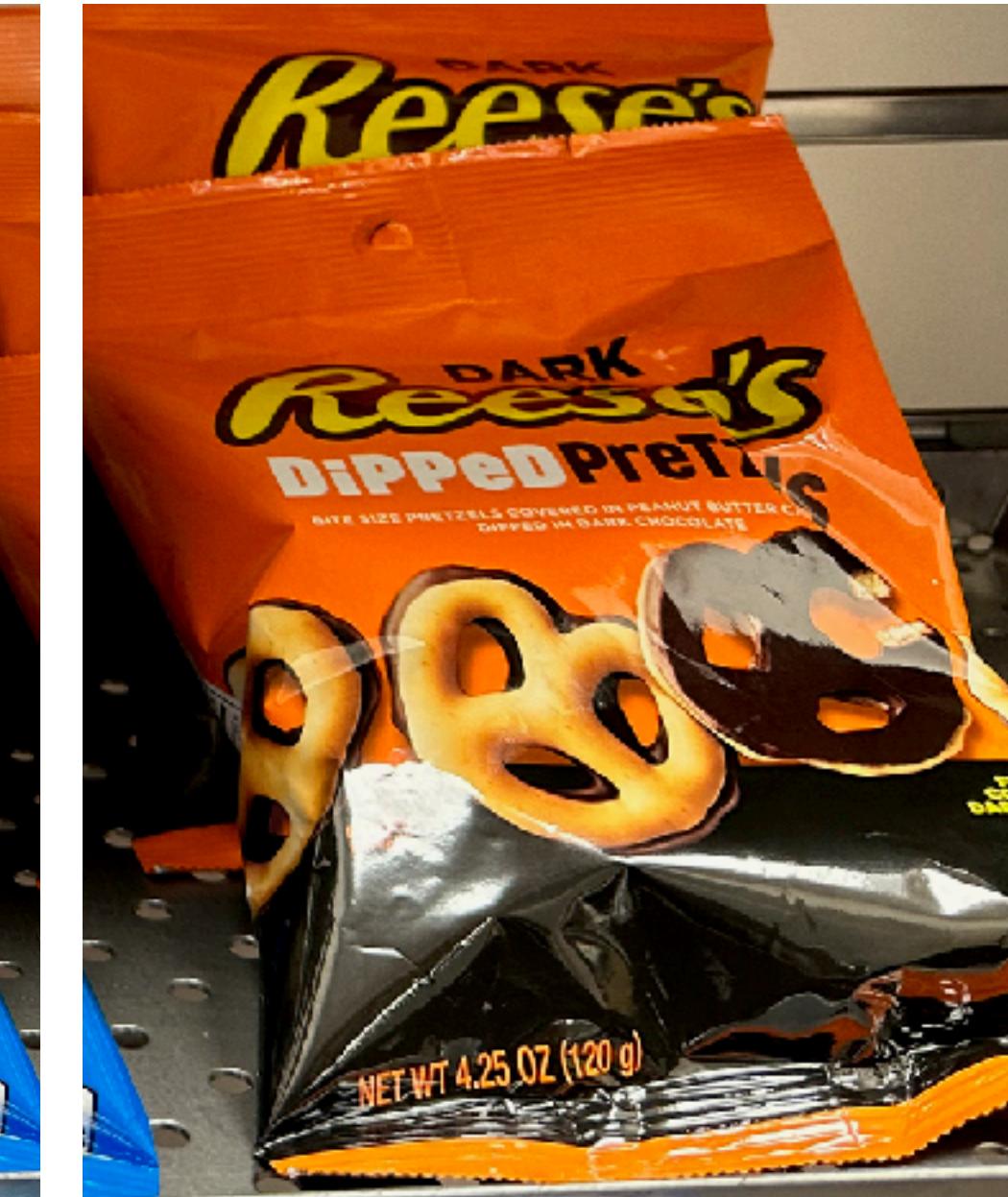
White  
Chocolate



Cookies N'  
Creme



Peanut Butter



Ambiguous  
Category



# Challenges—Background Clutter



# Challenges—Image Resolution

iPhone 14 Camera



4032x3024

ASUS RGB-D Camera



640x480

# Challenges—Illumination Changes



# Challenges—Illumination Changes



**Want our robot's perception system  
to be reliable in all conditions**

# Challenges—Subject Deformation

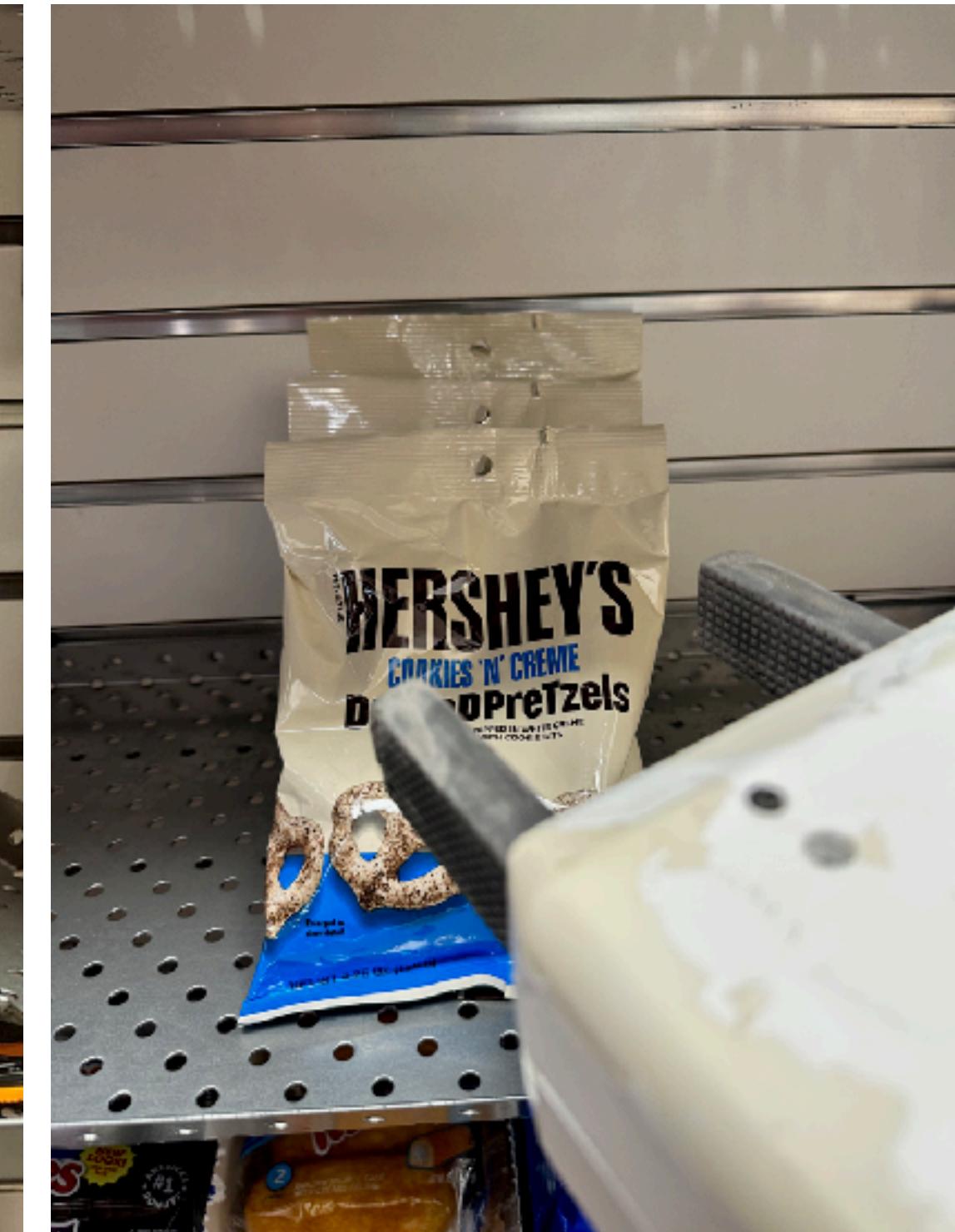


# Challenges—Occlusion

Scene Clutter



Robot Actuator



Transparency



# Challenges— Semantic Relationships

Reflections



Contact  
Relationships



# Challenges— Semantic Relationships

Reflections



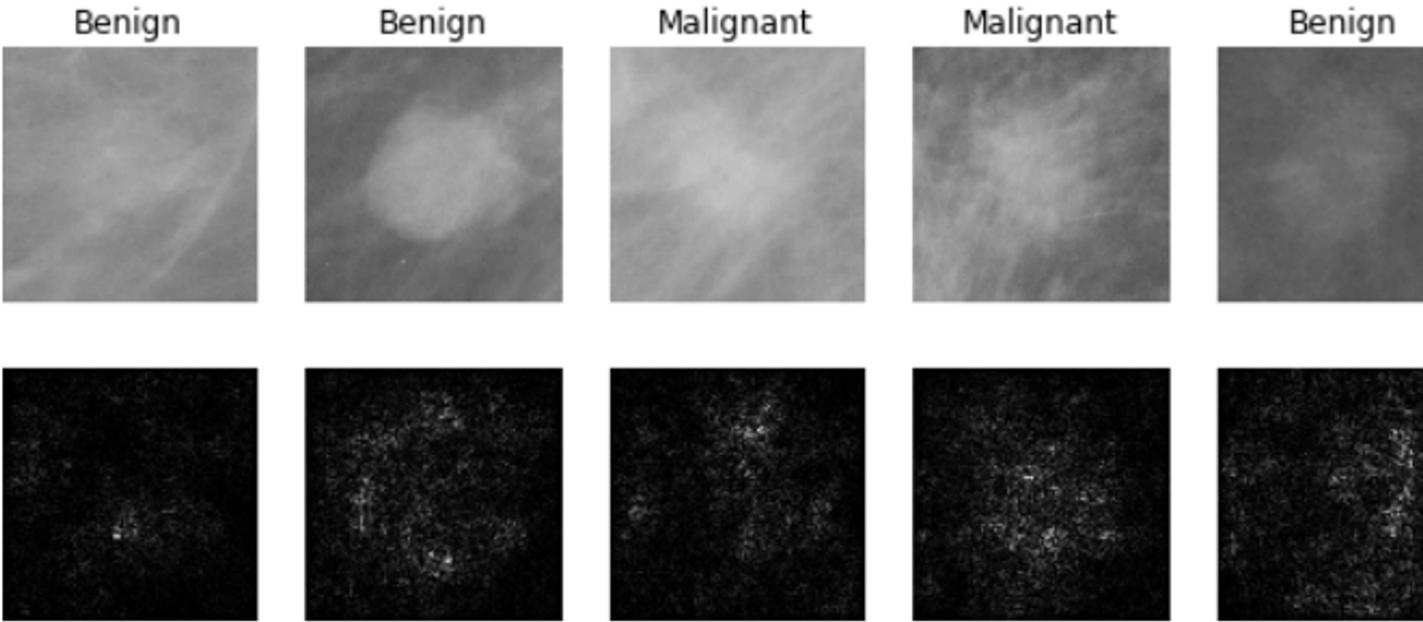
Contact  
Relationships



Robots have to act on the state they perceive

# Applications of Image Classification

## Medical Imaging



Lévy et al., "Breast Mass Classification from Mammograms using Deep Convolutional Neural Networks", arXiv:1612.00542, 2016

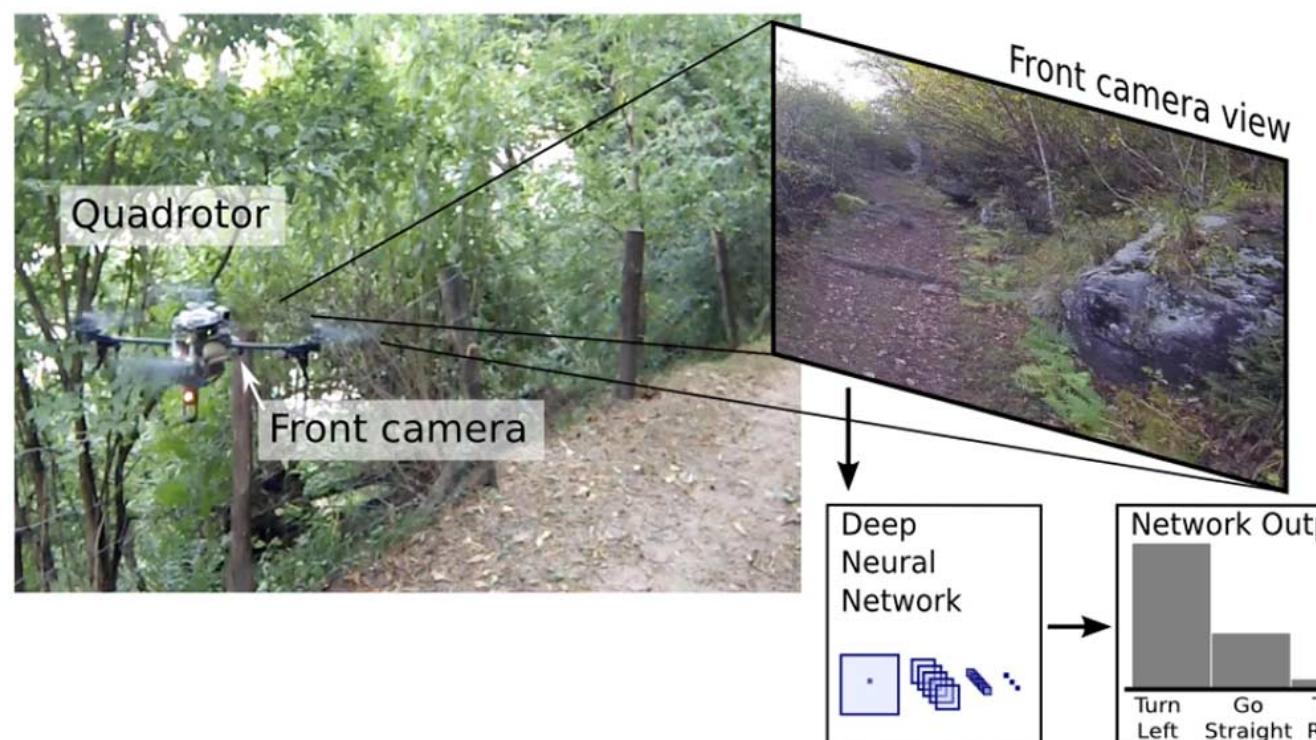
## Galaxy Classification



Dieleman et al., "Rotation-invariant convolutional neural networks for galaxy morphology prediction", 2015

From left to right: [public domain by NASA](#), [usage permitted by ESA/Hubble](#), [public domain by NASA](#), and [public domain](#)

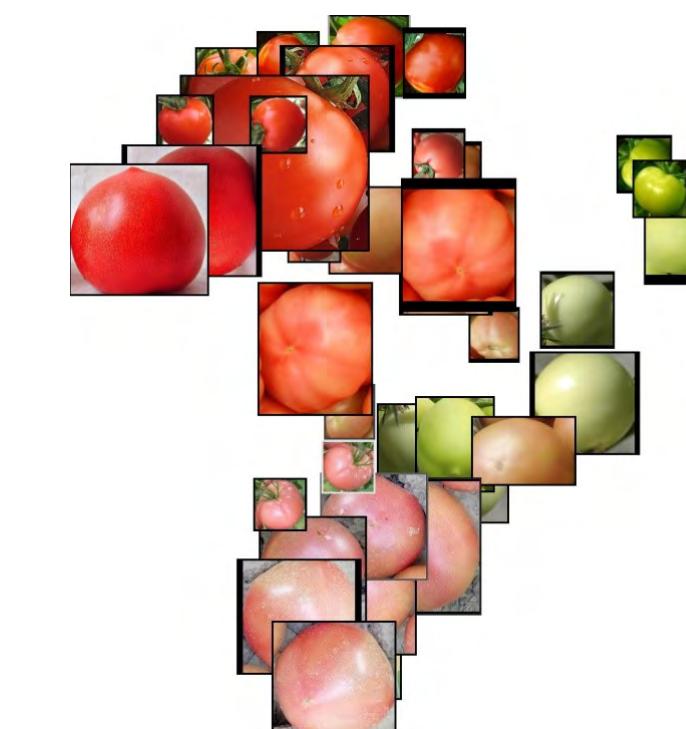
## Trail Direction Classification



Giusti et al., "A Machine Learning Approach to Visual Perception of Forest Trails for Mobile Robots", IEEE RAL, 2016

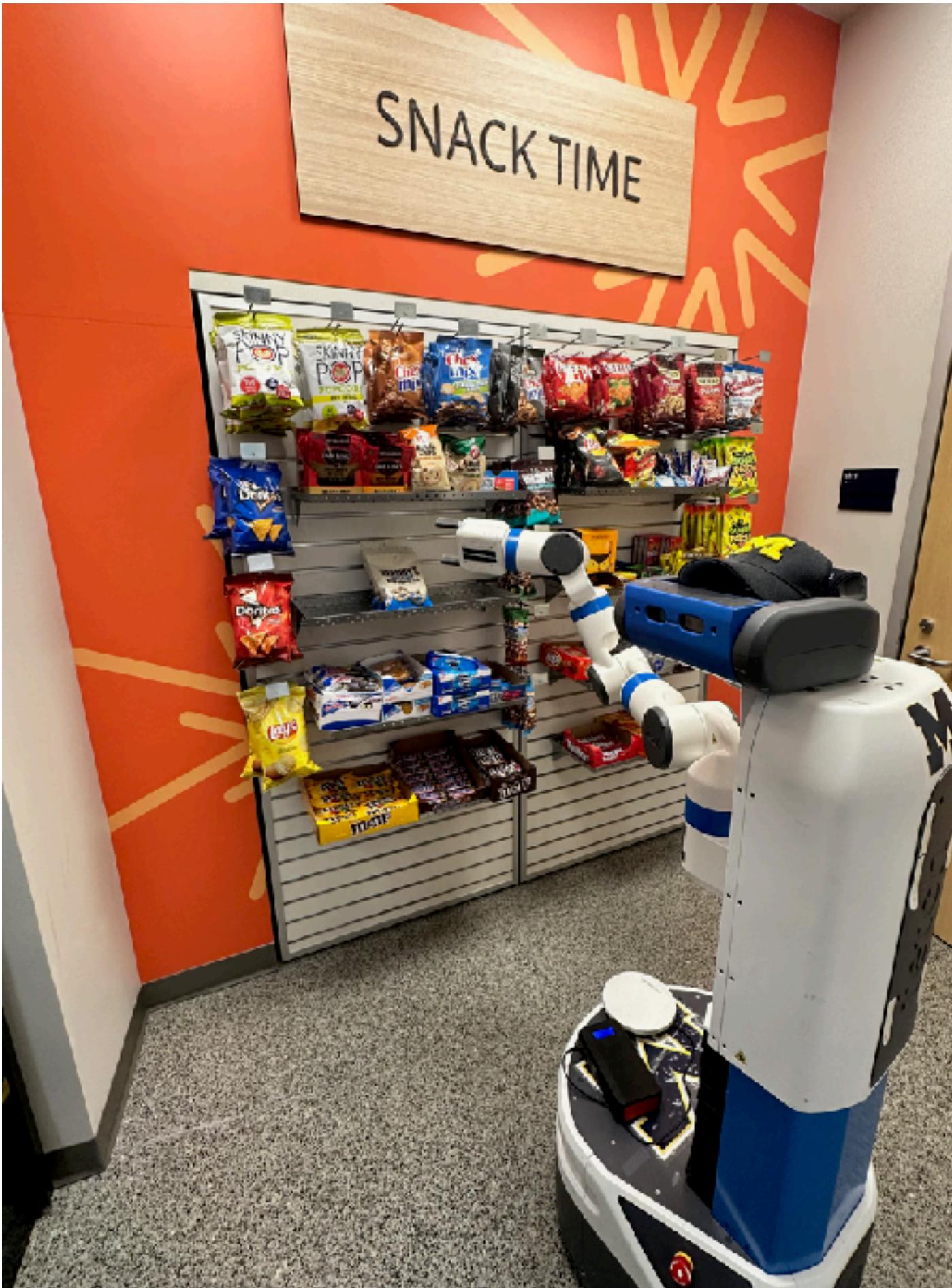
## Tomato Ripeness Classification

Name	Color	Storage Time (Days)	Sample
LV1	Breakers	21 ~ 28	
LV2	Turning	15 ~ 20	
LV3	Pink	7 ~ 14	
LV4	Light red	5 ~ 6	
LV5	Red	2 ~ 4	

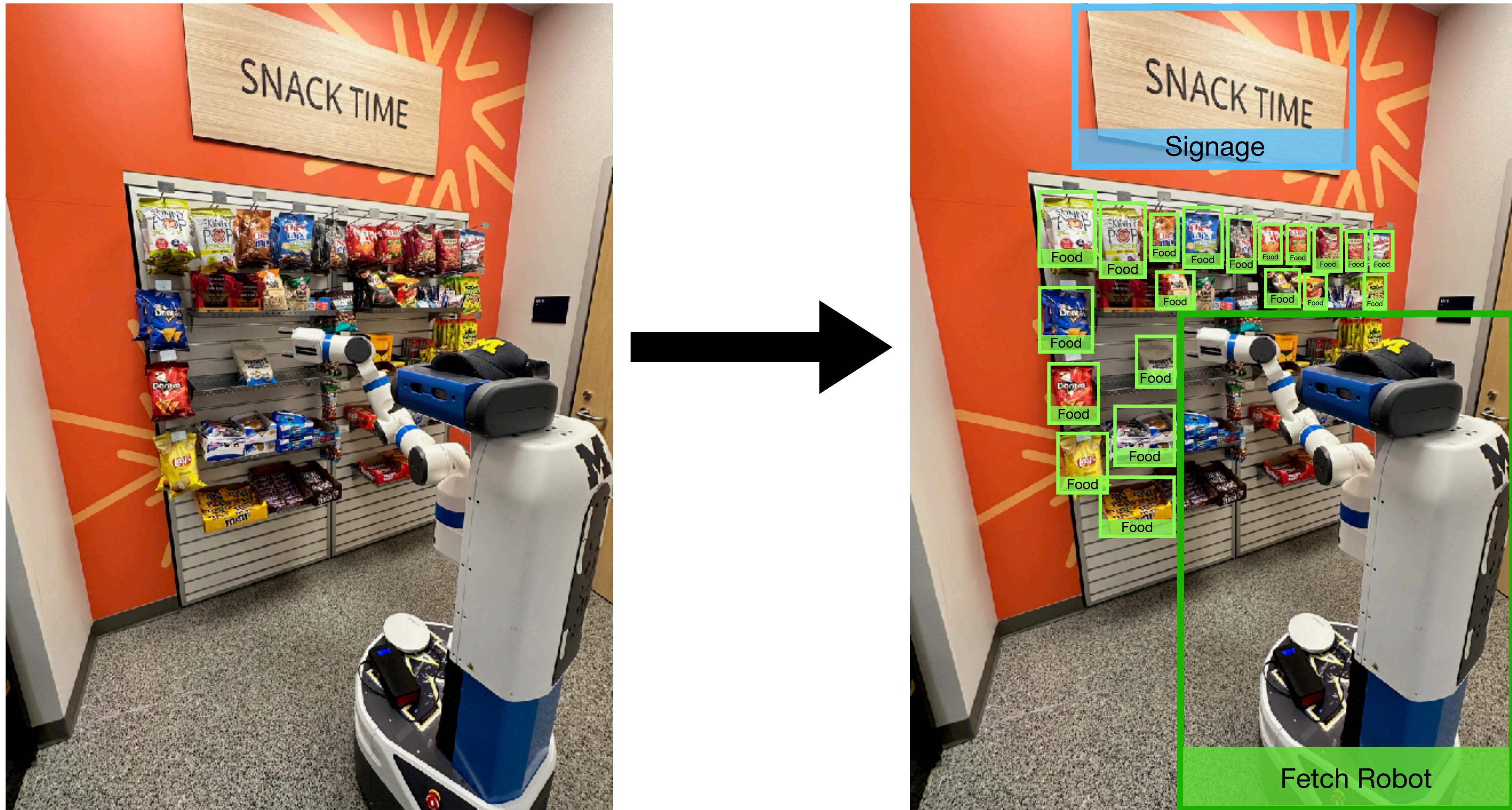


Zhang et al., "Deep Learning Based Improved Classification System for Designing Tomato Harvesting Robot", IEEE Access, 2016

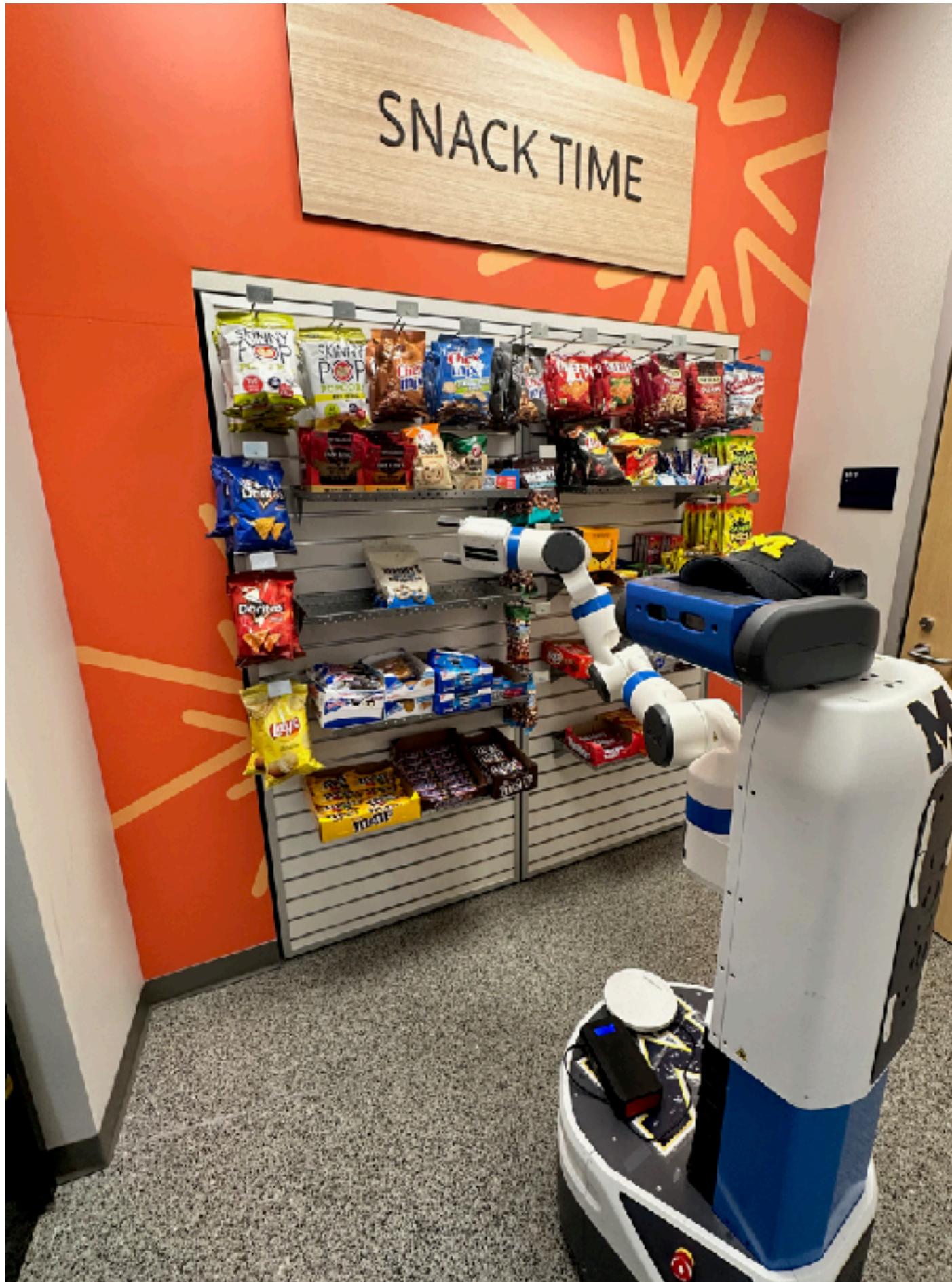
# Image Classification – Building Block for Other Tasks



# Image Classification – Building Block for Other Tasks



# Image Classification – Building Block for Other Tasks



**Example:** Object Detection

Wall

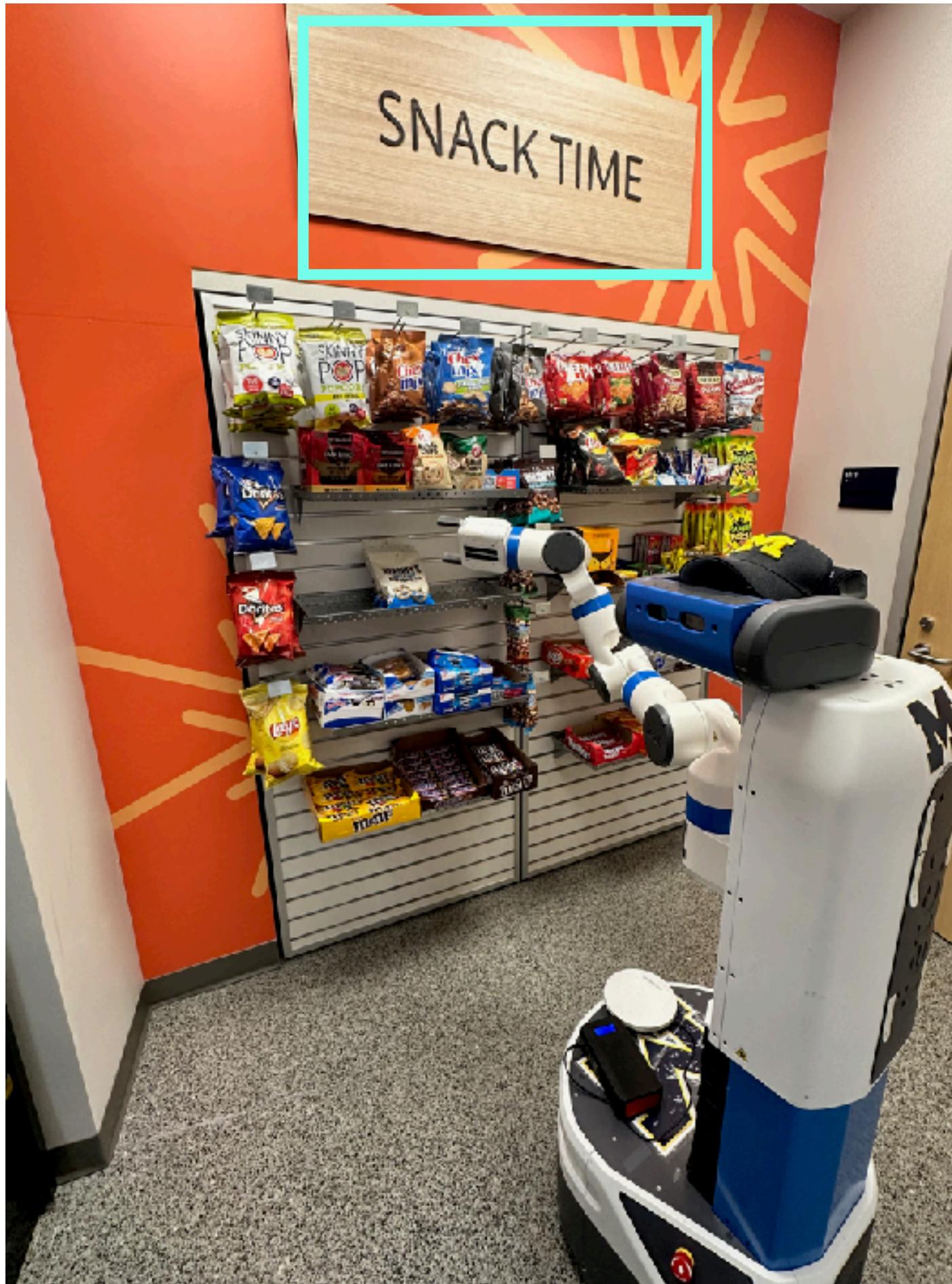
Floor

Signage

Fetch Robot

Snacks

# Image Classification – Building Block for Other Tasks



**Example:** Object Detection

Wall

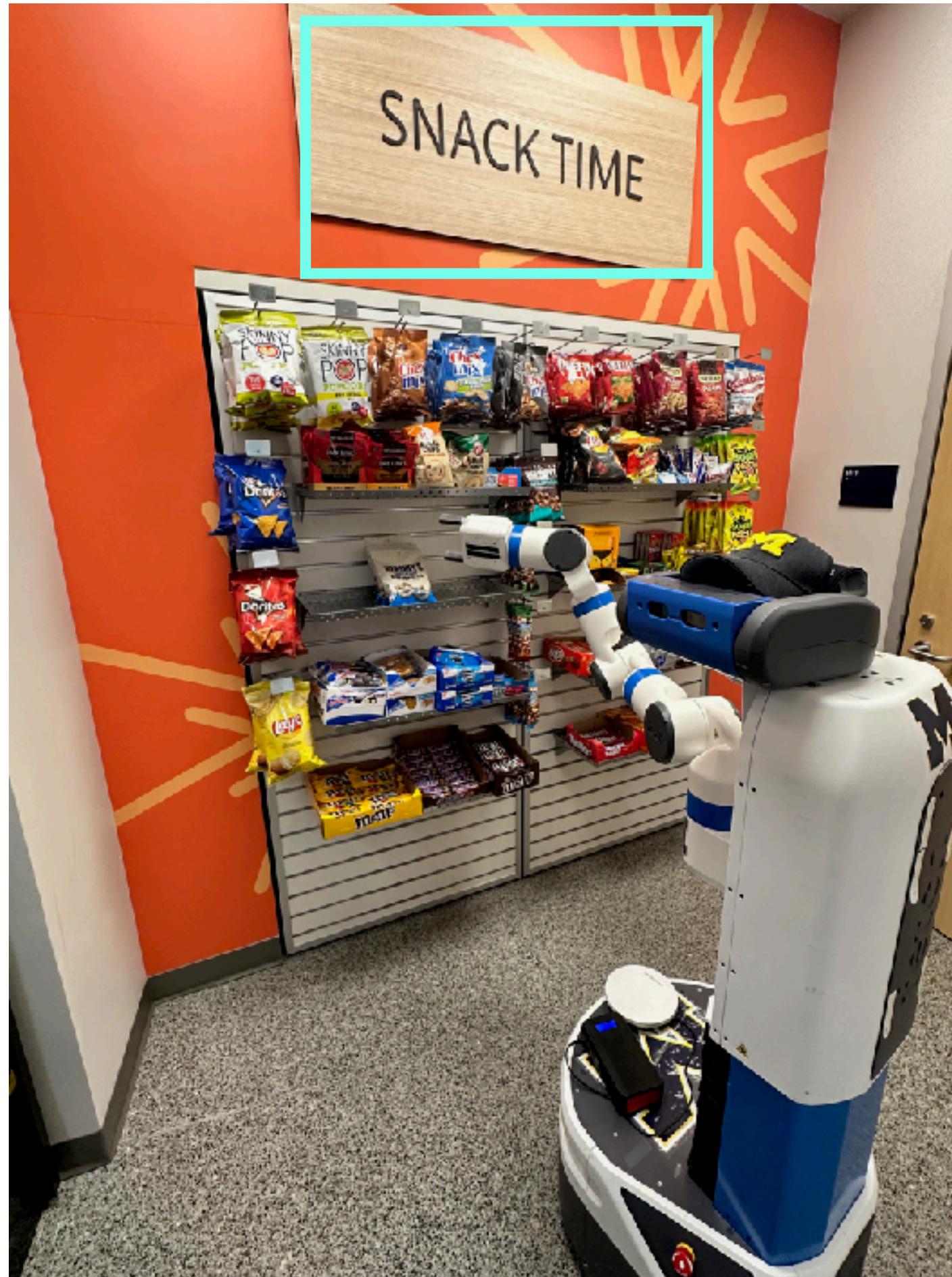
Floor

Signage

Fetch Robot

Snacks

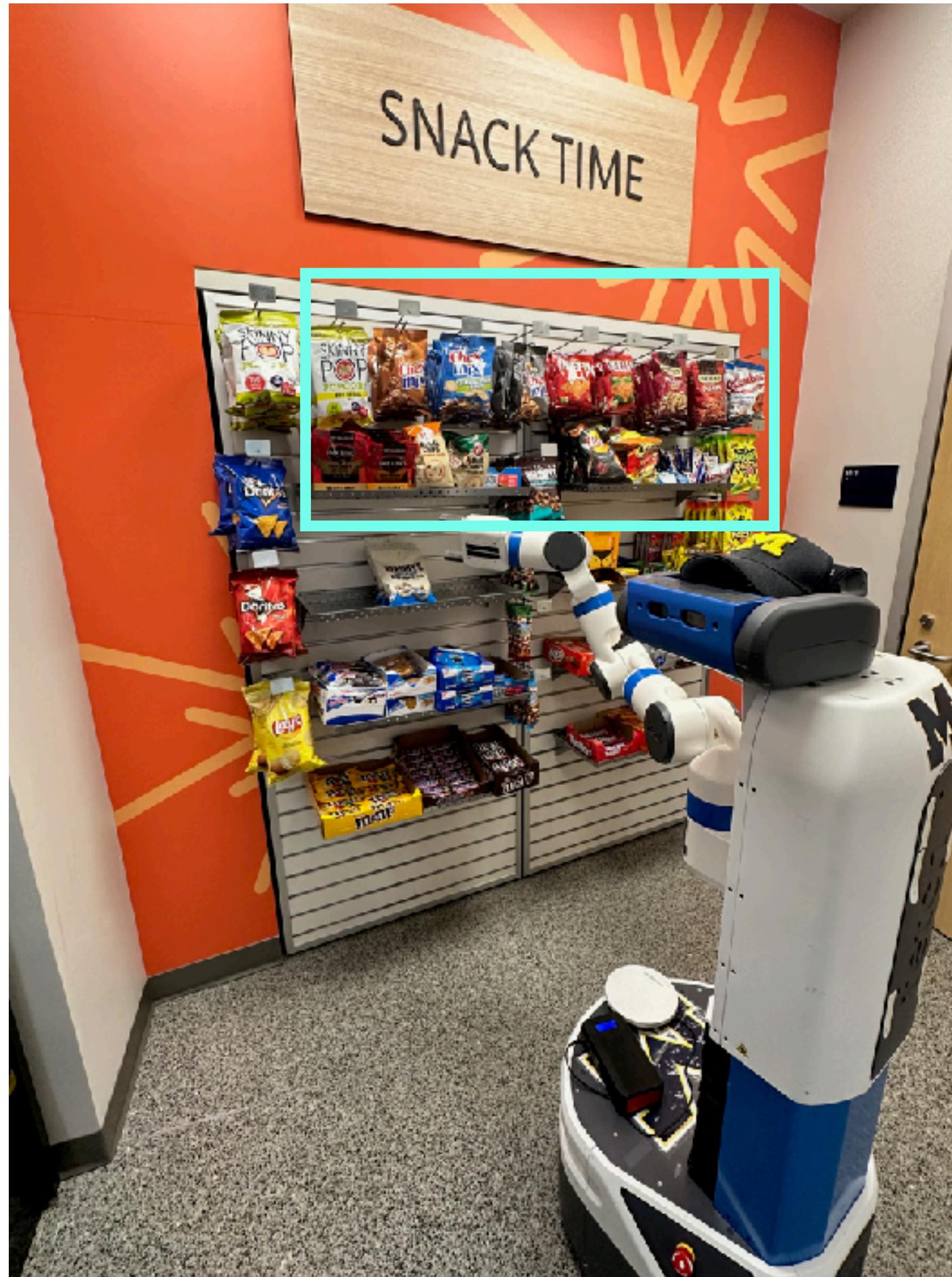
# Image Classification – Building Block for Other Tasks



**Example: Object Detection**

Wall  
Floor  
**Signage**  
Fetch Robot  
Snacks

# Image Classification – Building Block for Other Tasks



**Example: Object Detection**

Wall

Floor

Signage

Fetch Robot

Snacks

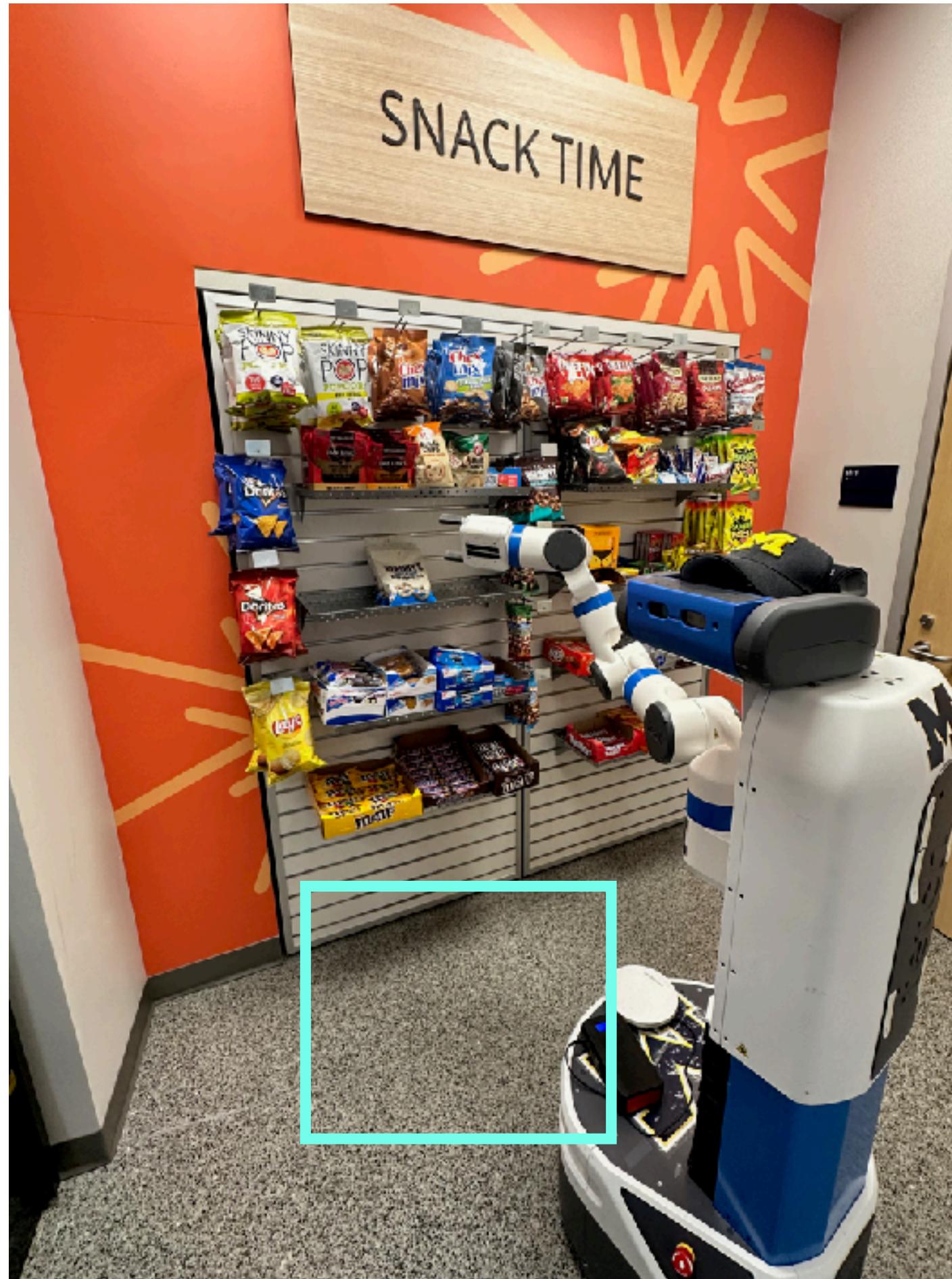
# Image Classification – Building Block for Other Tasks



**Example:** Object Detection

Wall  
Floor  
Signage  
**Fetch Robot**  
Snacks

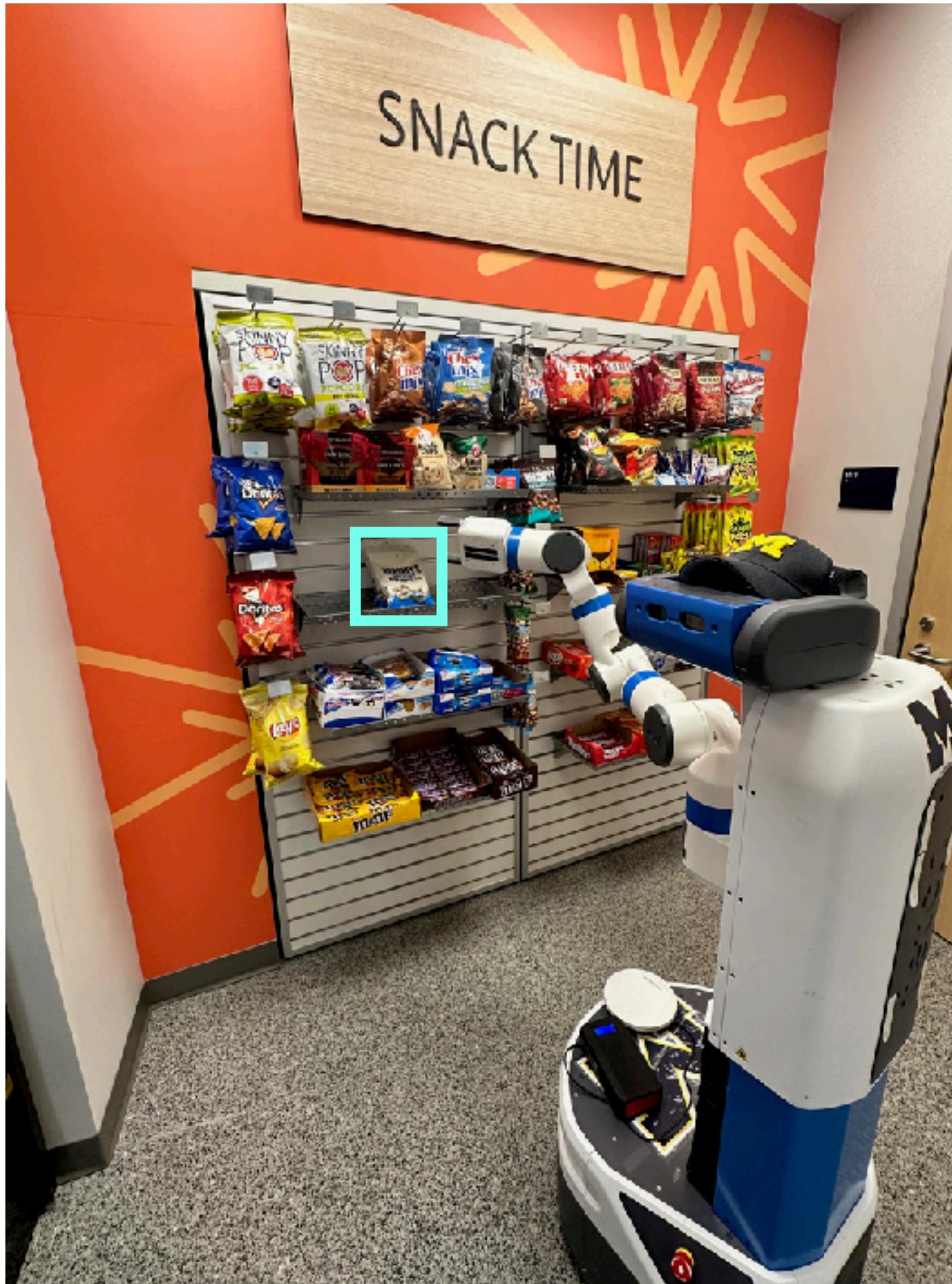
# Image Classification – Building Block for Other Tasks



**Example: Object Detection**

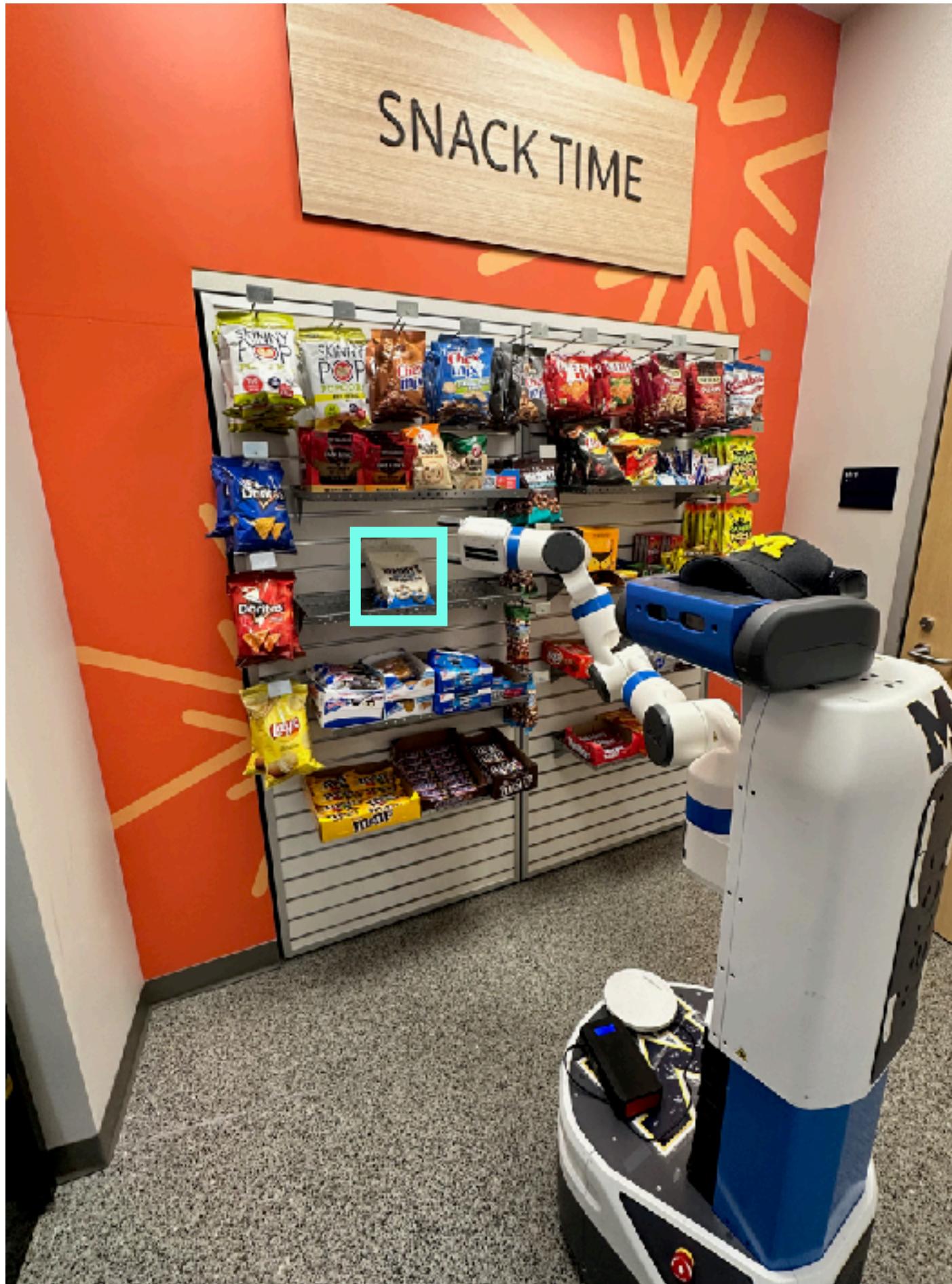
Wall  
Floor  
Signage  
Fetch Robot  
Snacks

# Image Classification – Building Block for Other Tasks



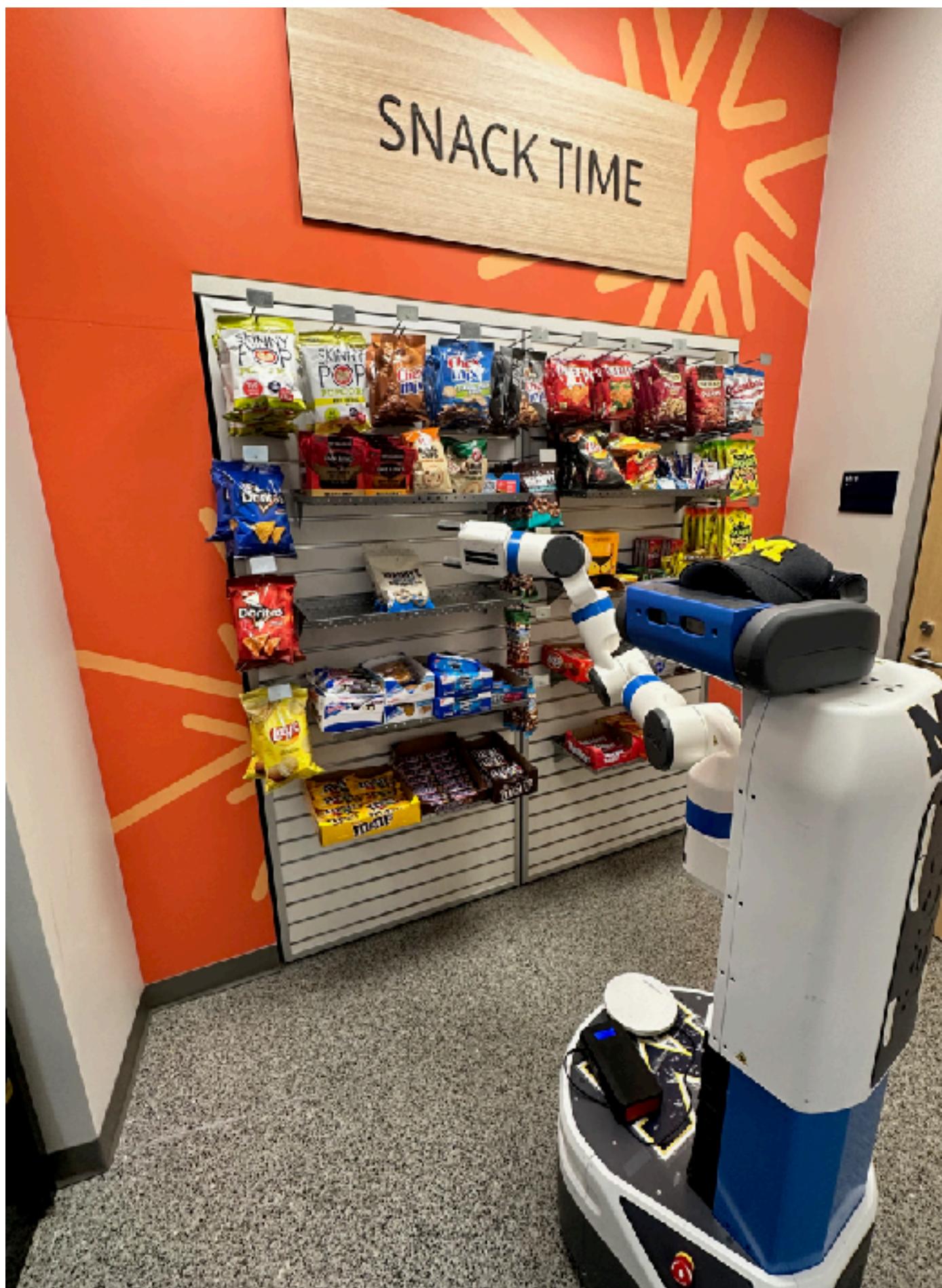
**Example:** Pose Estimation

# Image Classification – Building Block for Other Tasks



**Example:** Pose Estimation

# Image Classification – Building Block for Other Tasks



**Example:** Pose Estimation



# An Image Classifier

---

```
def classify_image(image):  
    # Some magic here?  
    return class_label
```



# An Image Classifier

---

```
def classify_image(image):  
    # Some magic here?  
    return class_label
```

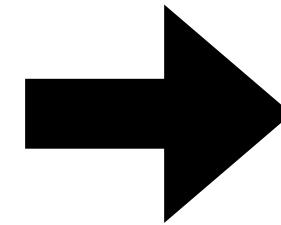
Unlike well defined programming (e.g. sorting a list)

No obvious way to hard-code the algorithm  
for recognizing each class

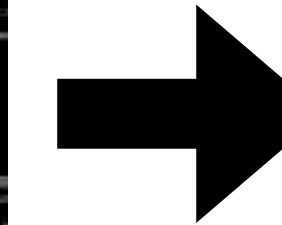


# An Image Classifier

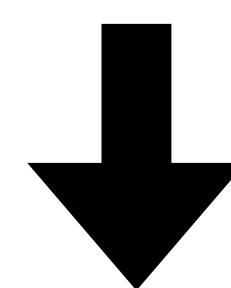
**Input:** image



**Detect:** Edges



**Detect:** Corners



???

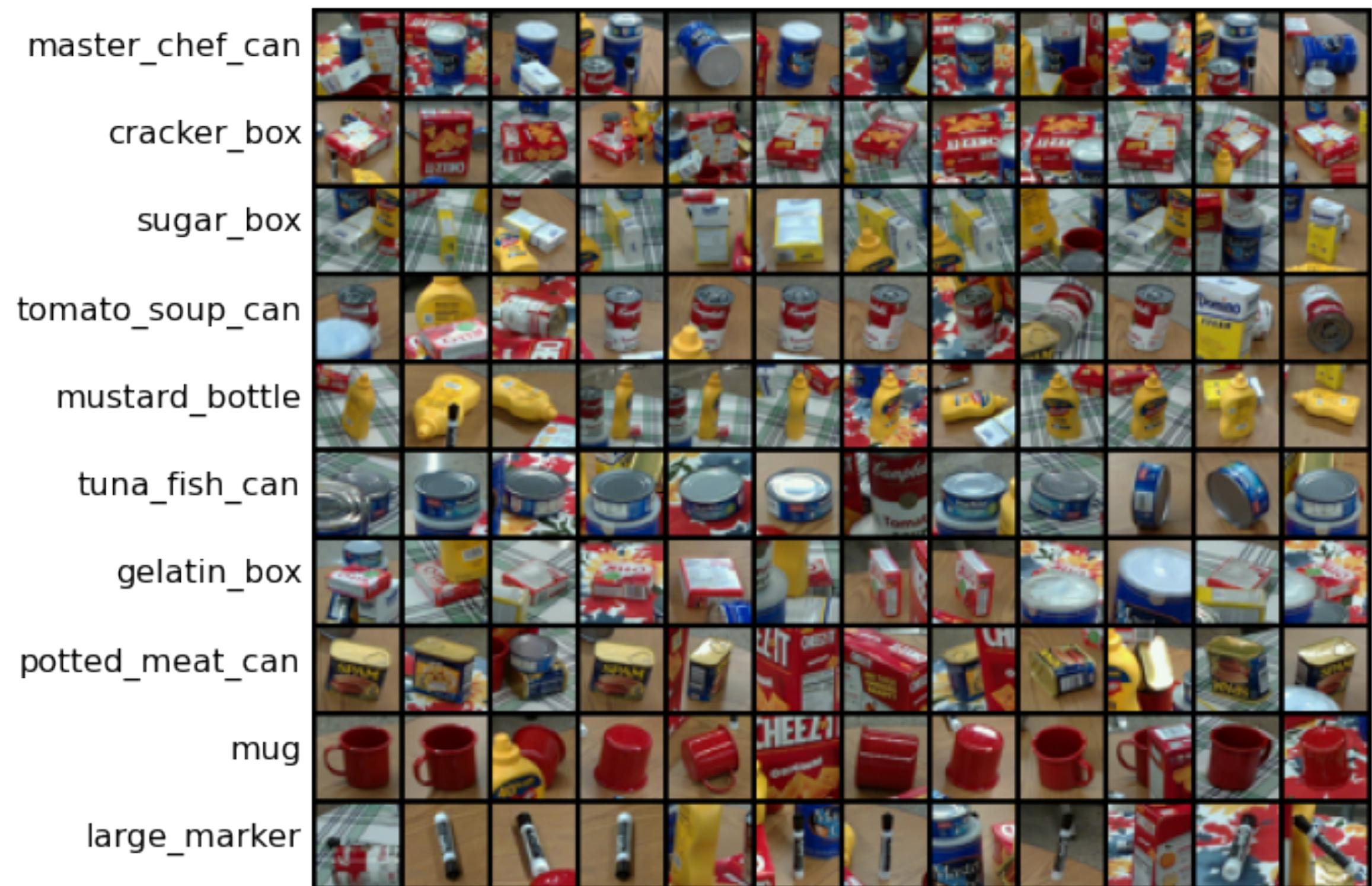
# Machine Learning—Data-Driven Approach

1. Collect a dataset of images and labels
2. Use Machine Learning to train a classifier
3. Evaluate the classifier on new images

```
def train(images, labels):
    # Machine learning!
    return model
```

```
def predict(model, test_images):
    # Use model to predict labels
    return test_labels
```

Example training set



# Image Classification Datasets—MNIST

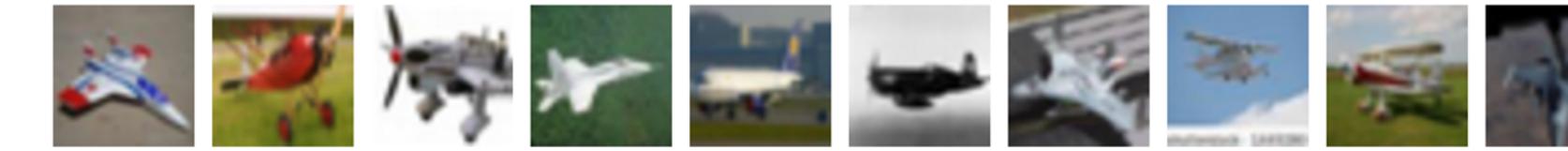


**10 classes:** Digits 0 to 9  
**28x28** grayscale images  
**50k** training images  
**10k** test images

Due to relatively small size,  
results on MNIST often do not  
hold on more complex datasets

# Image Classification Datasets—CIFAR10

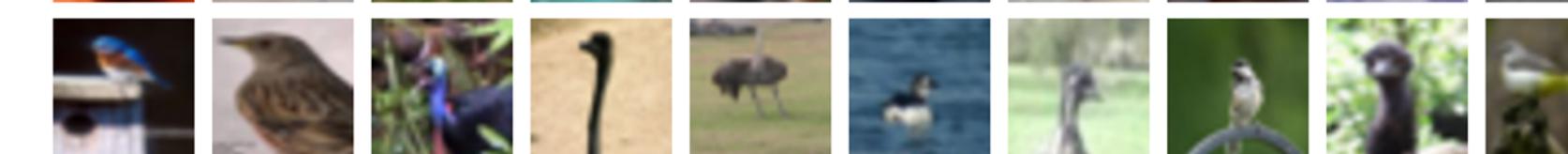
**airplane**



**automobile**



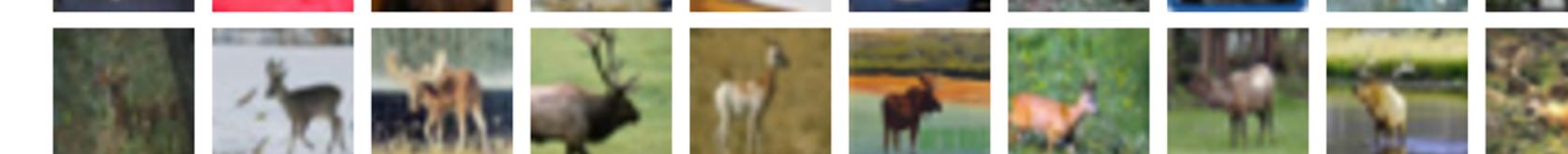
**bird**



**cat**



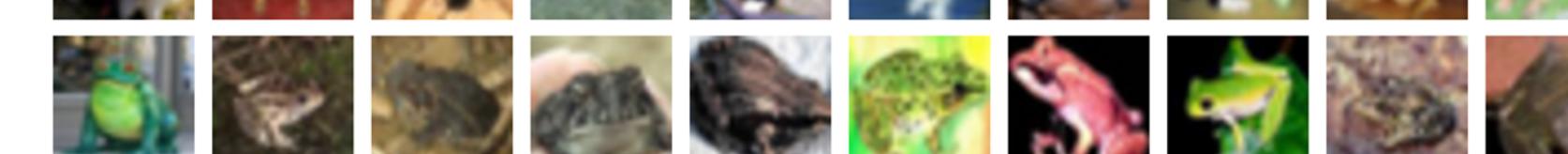
**deer**



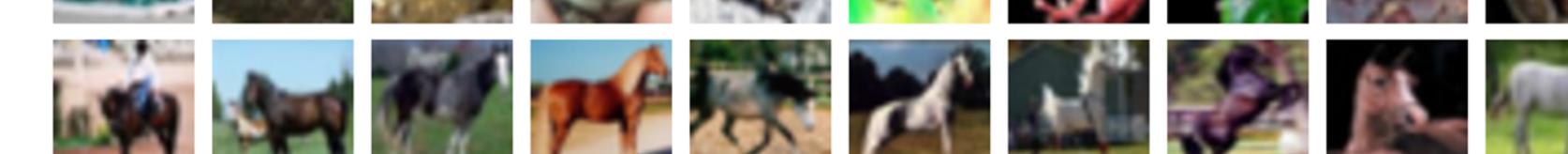
**dog**



**frog**



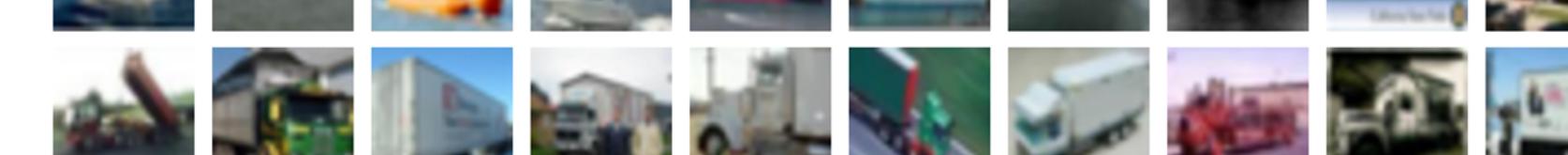
**horse**



**ship**



**truck**



**10 classes**

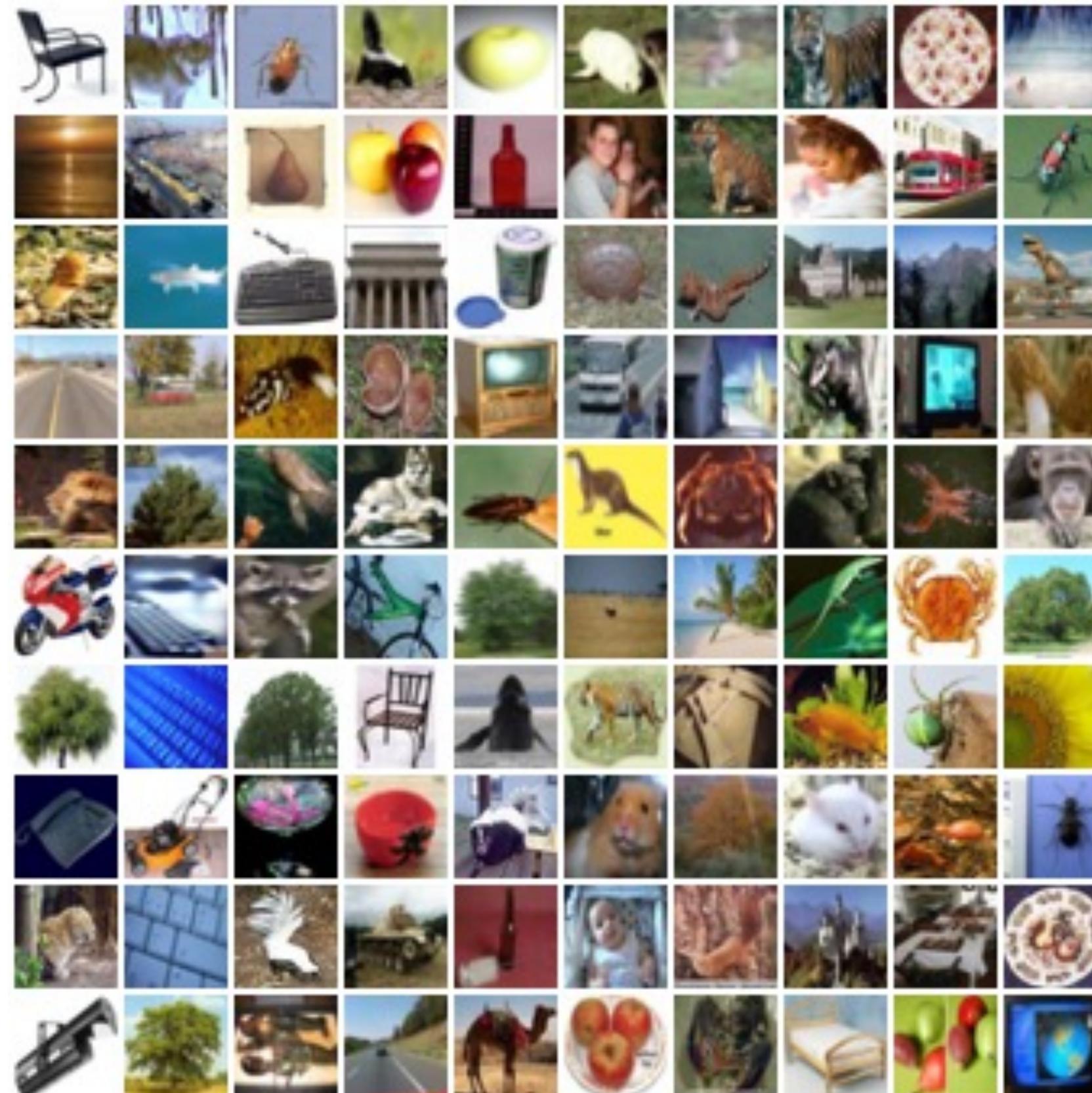
**32x32 RGB images**

**50k training images (5k per class)**

**10k test images (1k per class)**

Alex Krizhevsky, “Learning Multiple Layers of Features from Tiny Images”, Technical Report, 2009.

# Image Classification Datasets—CIFAR100



**100 classes**

**32x32 RGB images**

**50k training images (500 per class)**

**10k test images (100 per class)**

**20 superclasses** with 5 classes each:

Aquatic mammals: beaver, dolphin, otter, seal, whale

Trees: maple, oak, palm, pine, willow

Alex Krizhevsky, “Learning Multiple Layers of Features from Tiny Images”, Technical Report, 2009.

# Image Classification Datasets—ImageNet



**1000 classes**

**~1.3M** training images (~1.3K per class)  
**50k** validation images (50 per class)  
**100K** test images (100 per class)

Performance metric: **Top 5 accuracy**  
Algorithm predicts 5 labels for each image, one must be right

Deng et al., “ImageNet: A Large-Scale Hierarchical Image Database”, CVPR, 2009.  
Russakovsky et al., “ImageNet Large Scale Visual Recognition Challenge”, IJCV, 2015.

# Image Classification Datasets—ImageNet



Deng et al., "ImageNet: A Large-Scale Hierarchical Image Database", CVPR, 2009.  
Russakovsky et al., "ImageNet Large Scale Visual Recognition Challenge", IJCV, 2015.

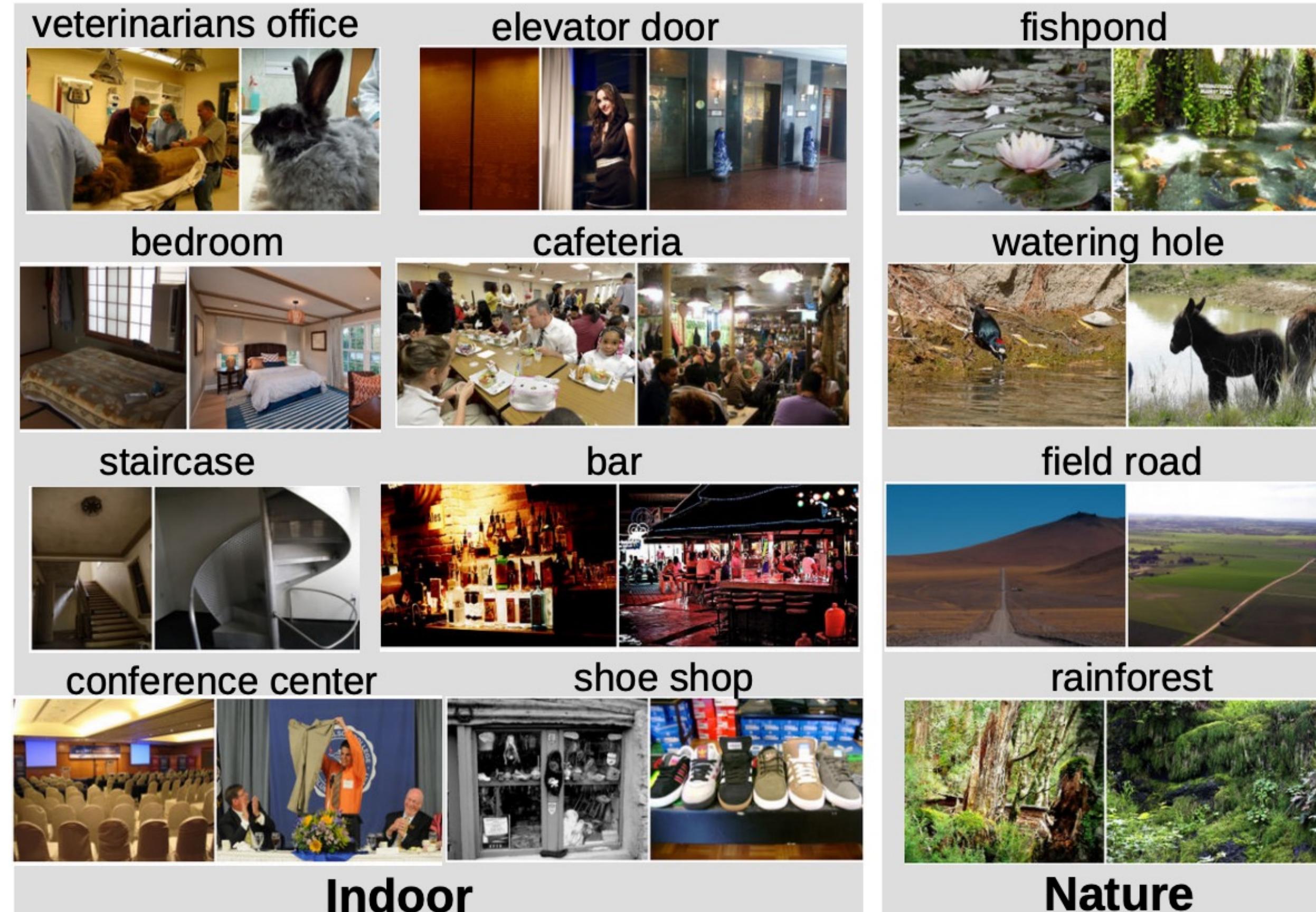
**1000 classes**

**~1.3M** training images (~1.3K per class)  
**50k** validation images (50 per class)  
**100K** test images (100 per class)  
test labels are secret!

Images have variable size, but often resized to **256x256** for training

There is also a **22K** category version of ImageNet, but less commonly used

# Image Classification Datasets—MIT Places



**365 classes** of different scene types

**~8M** training images

**18.25K** val images (50 per class)

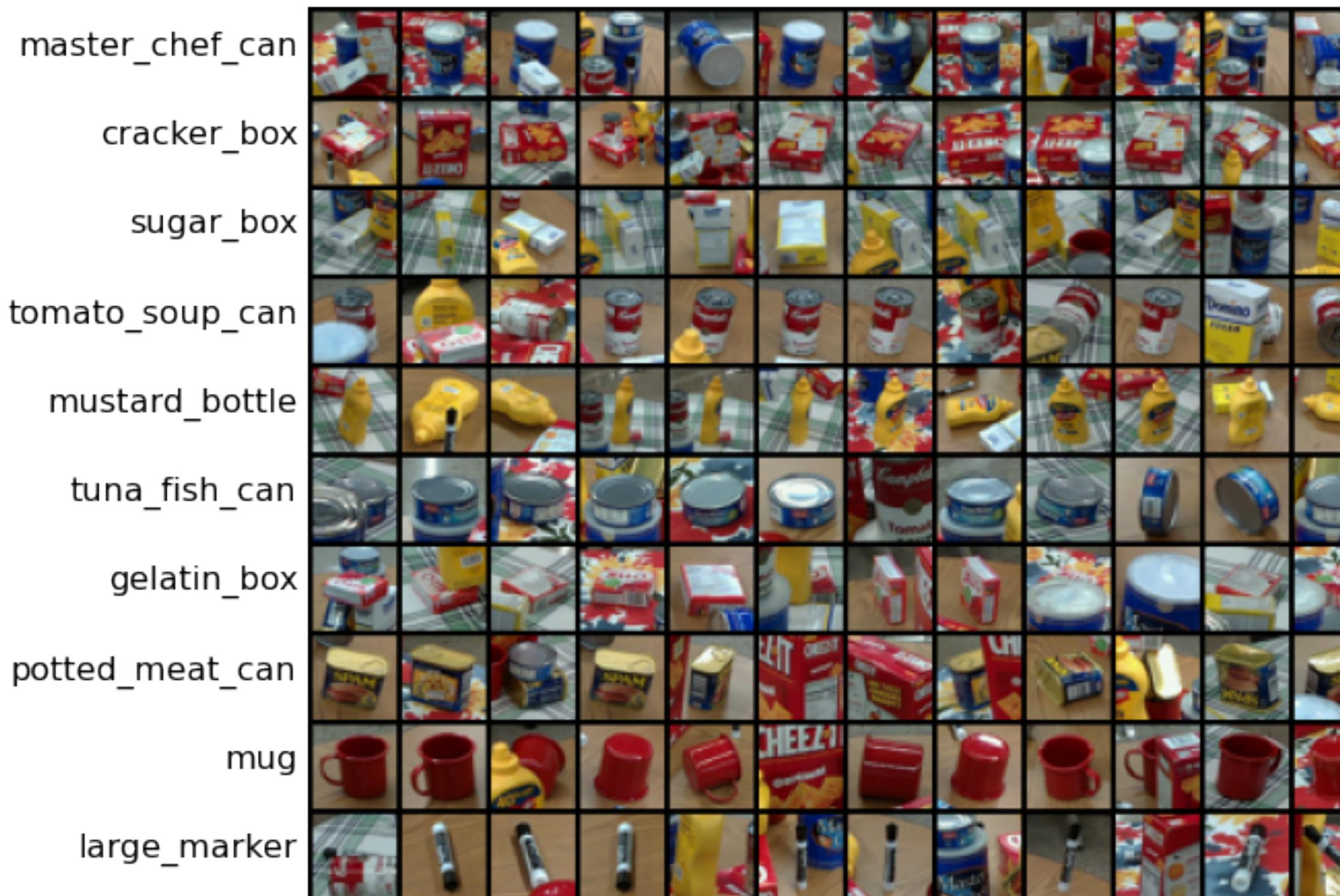
**328.5K** test images (900 per class)

Images have variable size, but often resized to **256x256** for training

Zhou et al., "Places: A 10 million Image Database for Scene Recognition", TPAMI, 2017.

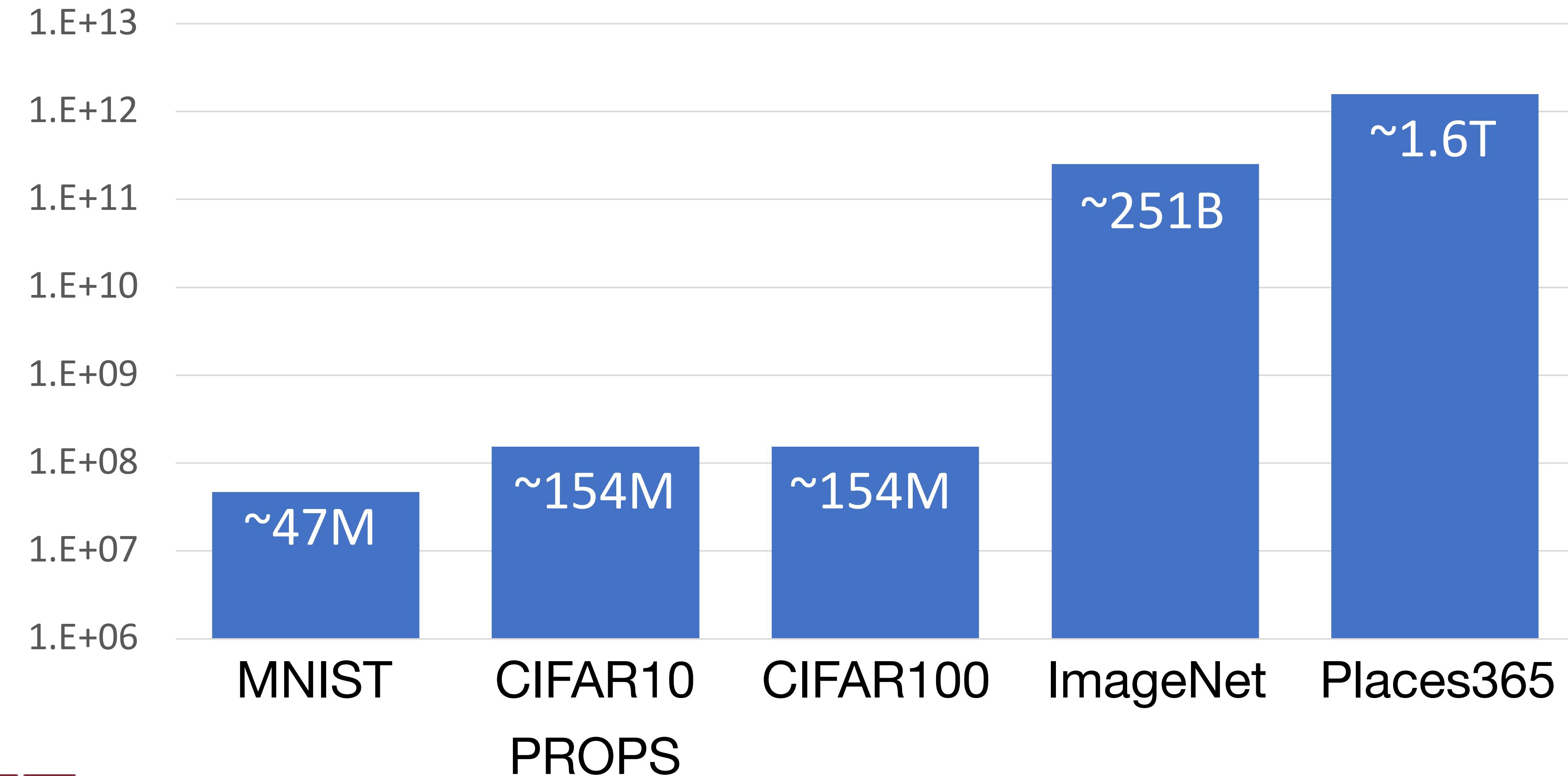
# Image Classification Datasets—PROPS

## Progress Robot Object Perception Samples Dataset



**10 classes**  
**32x32 RGB images**  
**50k training images (5k per class)**  
**10k test images (1k per class)**

# Classification Datasets—Number of Training Pixels



# First Classifier—Nearest Neighbor

```
def train(images, labels):  
    # Machine learning!  
    return model
```



Memorize all data and labels

```
def predict(model, test_images):  
    # Use model to predict labels  
    return test_labels
```



Predict the label of the most similar training image

# Distance Metric to Compare Images

**L1 distance:**  $d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$

test image	training image	pixel-wise absolute value differences	
56 32 10 18	10 20 24 17	46 12 14 1	
90 23 128 133	8 10 89 100	82 13 39 33	
24 26 178 200	12 16 178 170	12 10 0 30	
2 0 255 220	4 32 233 112	2 32 22 108	
-		= add → 456	



# Nearest Neighbor Classifier

```
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        """ X is N x D where each row is an example. Y is 1-dimension of size N """
        # the nearest neighbor classifier simply remembers all the training data
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        """ X is N x D where each row is an example we wish to predict label for """
        num_test = X.shape[0]
        # lets make sure that the output type matches the input type
        Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

        # loop over all test rows
        for i in xrange(num_test):
            # find the nearest training image to the i'th test image
            # using the L1 distance (sum of absolute value differences)
            distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
            min_index = np.argmin(distances) # get the index with smallest distance
            Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

        return Ypred
```



# Nearest Neighbor Classifier

```
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        """ X is N x D where each row is an example. Y is 1-dimension of size N """
        # the nearest neighbor classifier simply remembers all the training data
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        """ X is N x D where each row is an example we wish to predict label for """
        num_test = X.shape[0]
        # lets make sure that the output type matches the input type
        Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

        # loop over all test rows
        for i in xrange(num_test):
            # find the nearest training image to the i'th test image
            # using the L1 distance (sum of absolute value differences)
            distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
            min_index = np.argmin(distances) # get the index with smallest distance
            Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

        return Ypred
```

Memorize training data



# Nearest Neighbor Classifier

```
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        """ X is N x D where each row is an example. Y is 1-dimension of size N """
        # the nearest neighbor classifier simply remembers all the training data
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        """ X is N x D where each row is an example we wish to predict label for """
        num_test = X.shape[0]
        # lets make sure that the output type matches the input type
        Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

        # loop over all test rows
        for i in xrange(num_test):
            # find the nearest training image to the i'th test image
            # using the L1 distance (sum of absolute value differences)
            distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
            min_index = np.argmin(distances) # get the index with smallest distance
            Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

        return Ypred
```

For each test image:  
Find nearest training image  
Return label of nearest image



# Nearest Neighbor Classifier

```
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        """ X is N x D where each row is an example. Y is 1-dimension of size N """
        # the nearest neighbor classifier simply remembers all the training data
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        """ X is N x D where each row is an example we wish to predict label for """
        num_test = X.shape[0]
        # lets make sure that the output type matches the input type
        Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

        # loop over all test rows
        for i in xrange(num_test):
            # find the nearest training image to the i'th test image
            # using the L1 distance (sum of absolute value differences)
            distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
            min_index = np.argmin(distances) # get the index with smallest distance
            Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

        return Ypred
```

Q: With N examples how fast is training?

A: O(1)



# Nearest Neighbor Classifier

```
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        """ X is N x D where each row is an example. Y is 1-dimension of size N """
        # the nearest neighbor classifier simply remembers all the training data
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        """ X is N x D where each row is an example we wish to predict label for """
        num_test = X.shape[0]
        # lets make sure that the output type matches the input type
        Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

        # loop over all test rows
        for i in xrange(num_test):
            # find the nearest training image to the i'th test image
            # using the L1 distance (sum of absolute value differences)
            distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
            min_index = np.argmin(distances) # get the index with smallest distance
            Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

        return Ypred
```

Q: With N examples how fast is training?

A: O(1)

Q: With N examples how fast is testing?

A: O(N)



# Nearest Neighbor Classifier

```
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        """ X is N x D where each row is an example. Y is 1-dimension of size N """
        # the nearest neighbor classifier simply remembers all the training data
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        """ X is N x D where each row is an example we wish to predict label for """
        num_test = X.shape[0]
        # lets make sure that the output type matches the input type
        Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

        # loop over all test rows
        for i in xrange(num_test):
            # find the nearest training image to the i'th test image
            # using the L1 distance (sum of absolute value differences)
            distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
            min_index = np.argmin(distances) # get the index with smallest distance
            Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

        return Ypred
```

Q: With N examples how fast is training?

A: O(1)

Q: With N examples how fast is testing?

A: O(N)

This is a problem: we can train slow offline but need fast testing!



# Nearest Neighbor Classifier

```
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        """ X is N x D where each row is an example. Y is 1-dimension of size N """
        # the nearest neighbor classifier simply remembers all the training data
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        """ X is N x D where each row is an example we wish to predict label for """
        num_test = X.shape[0]
        # lets make sure that the output type matches the input type
        Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

        # loop over all test rows
        for i in xrange(num_test):
            # find the nearest training image to the i'th test image
            # using the L1 distance (sum of absolute value differences)
            distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
            min_index = np.argmin(distances) # get the index with smallest distance
            Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

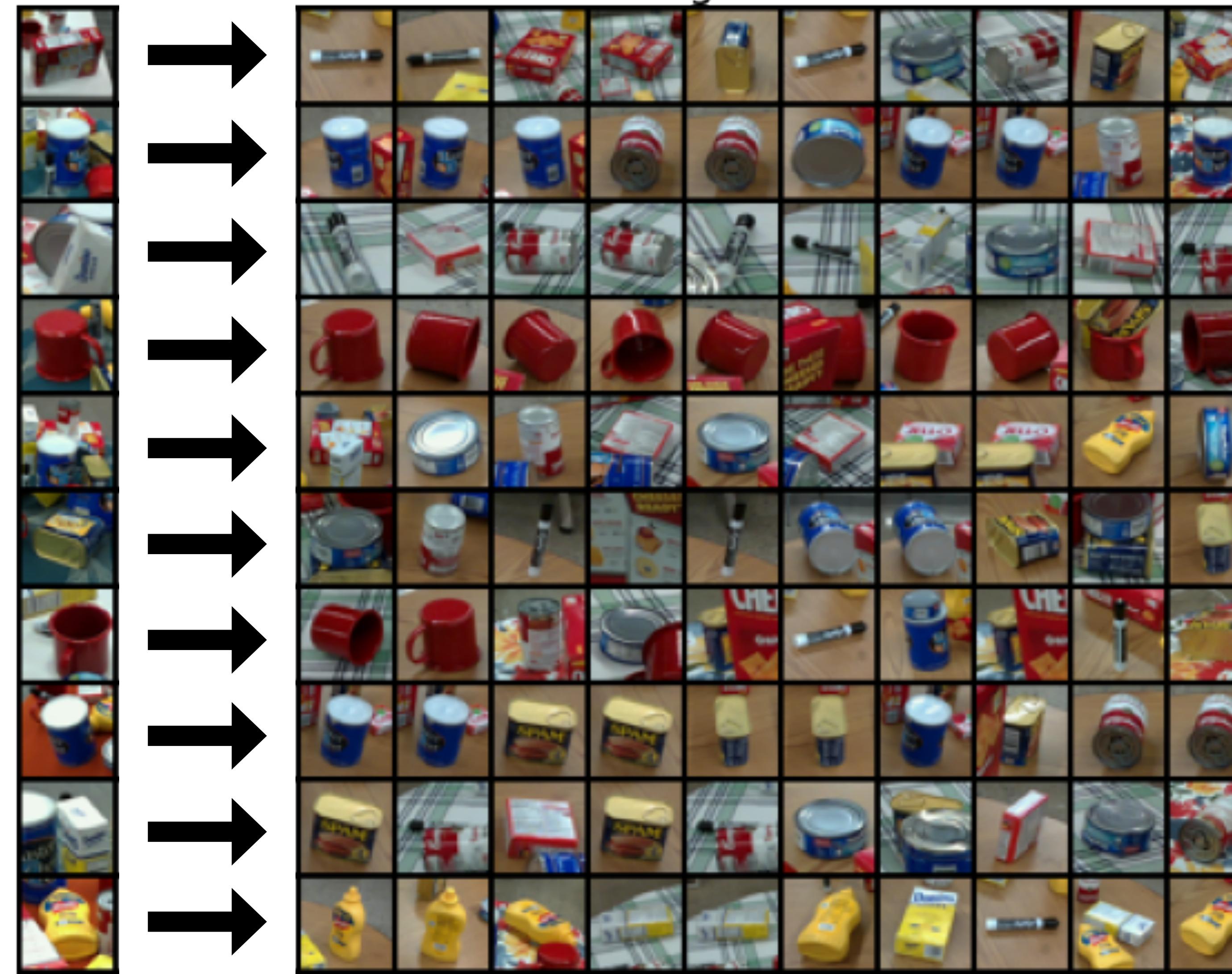
        return Ypred
```

There are many methods  
for fast / approximate  
nearest neighbors

e.g. [github.com/facebookresearch/faiss](https://github.com/facebookresearch/faiss)



# What does this look like?

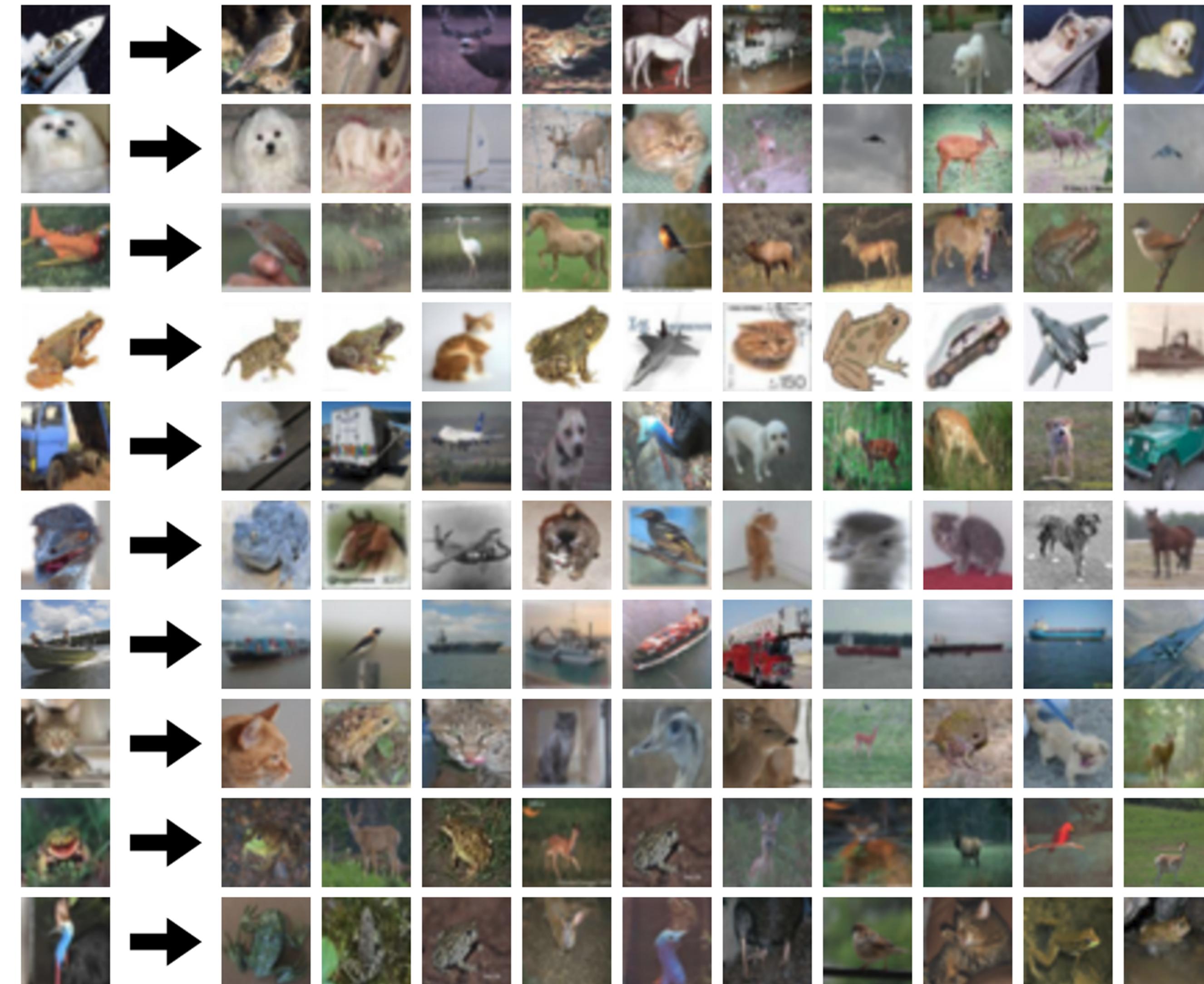


# What does this look like?

PROPS dataset is  
instance-level



# What does this look like?



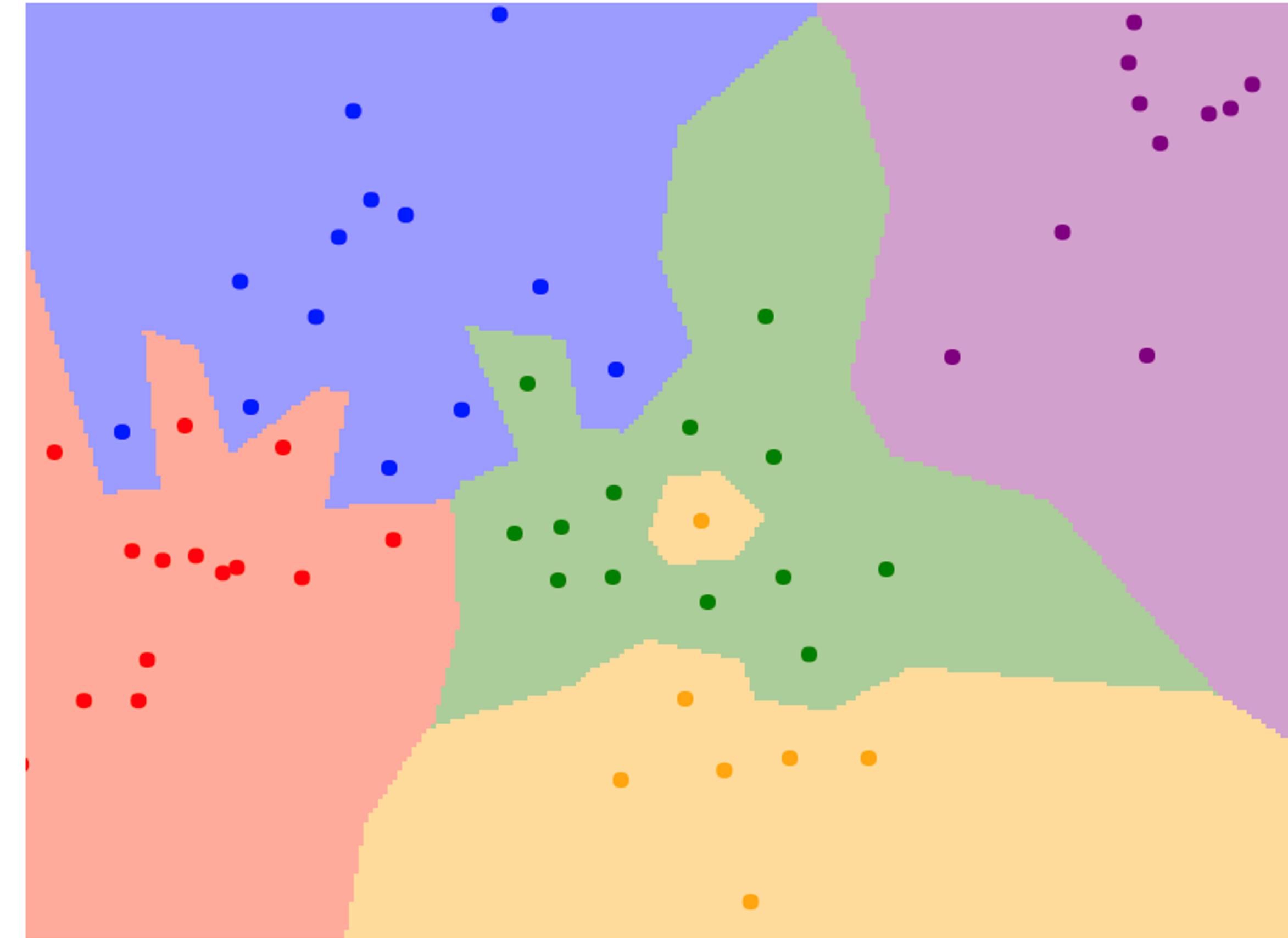
# What does this look like?

CIFAR10 dataset is  
category-level



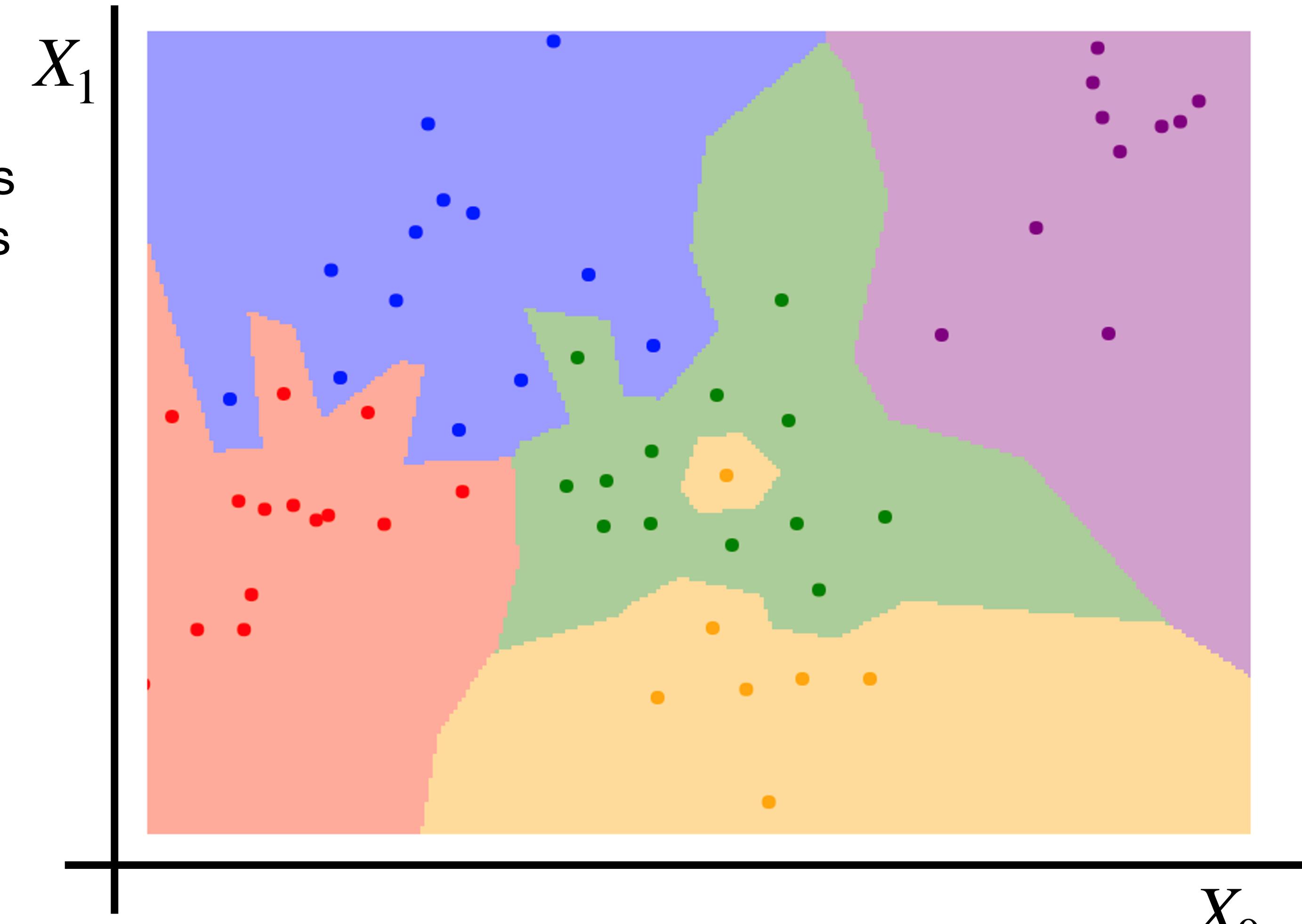
# K-Nearest Neighbors Decision Boundaries

---

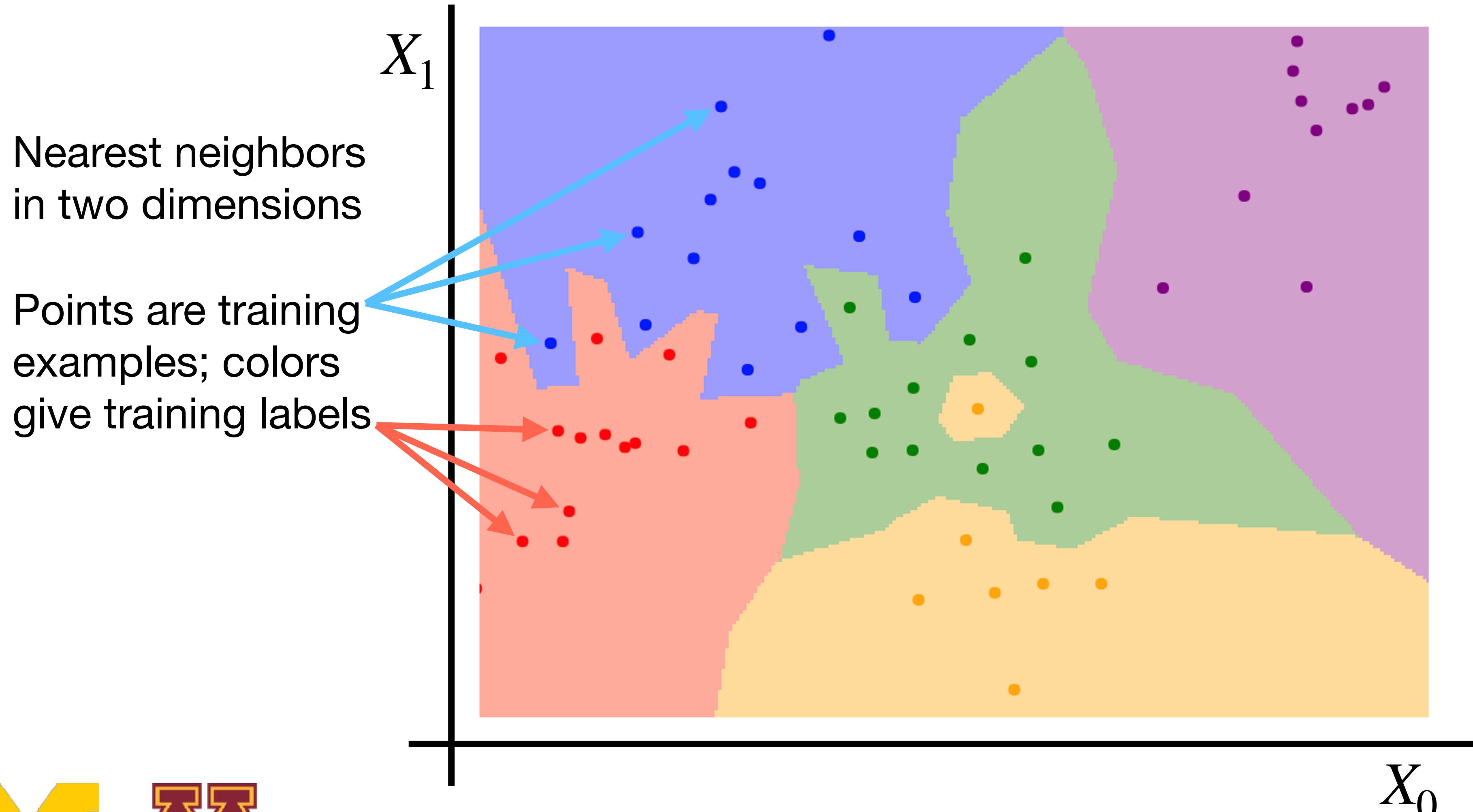


# K-Nearest Neighbors Decision Boundaries

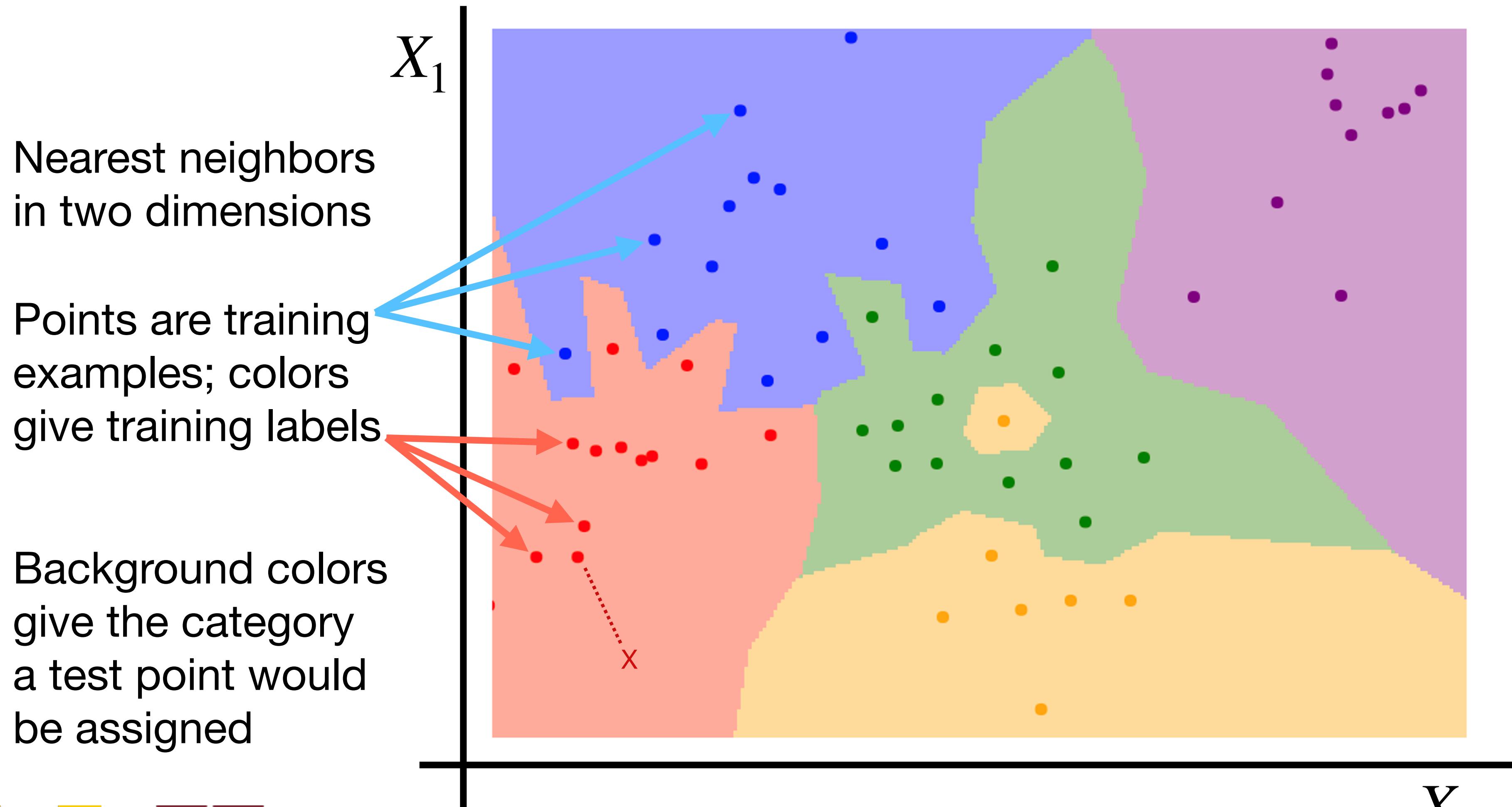
Nearest neighbors  
in two dimensions



# K-Nearest Neighbors Decision Boundaries



# K-Nearest Neighbors Decision Boundaries

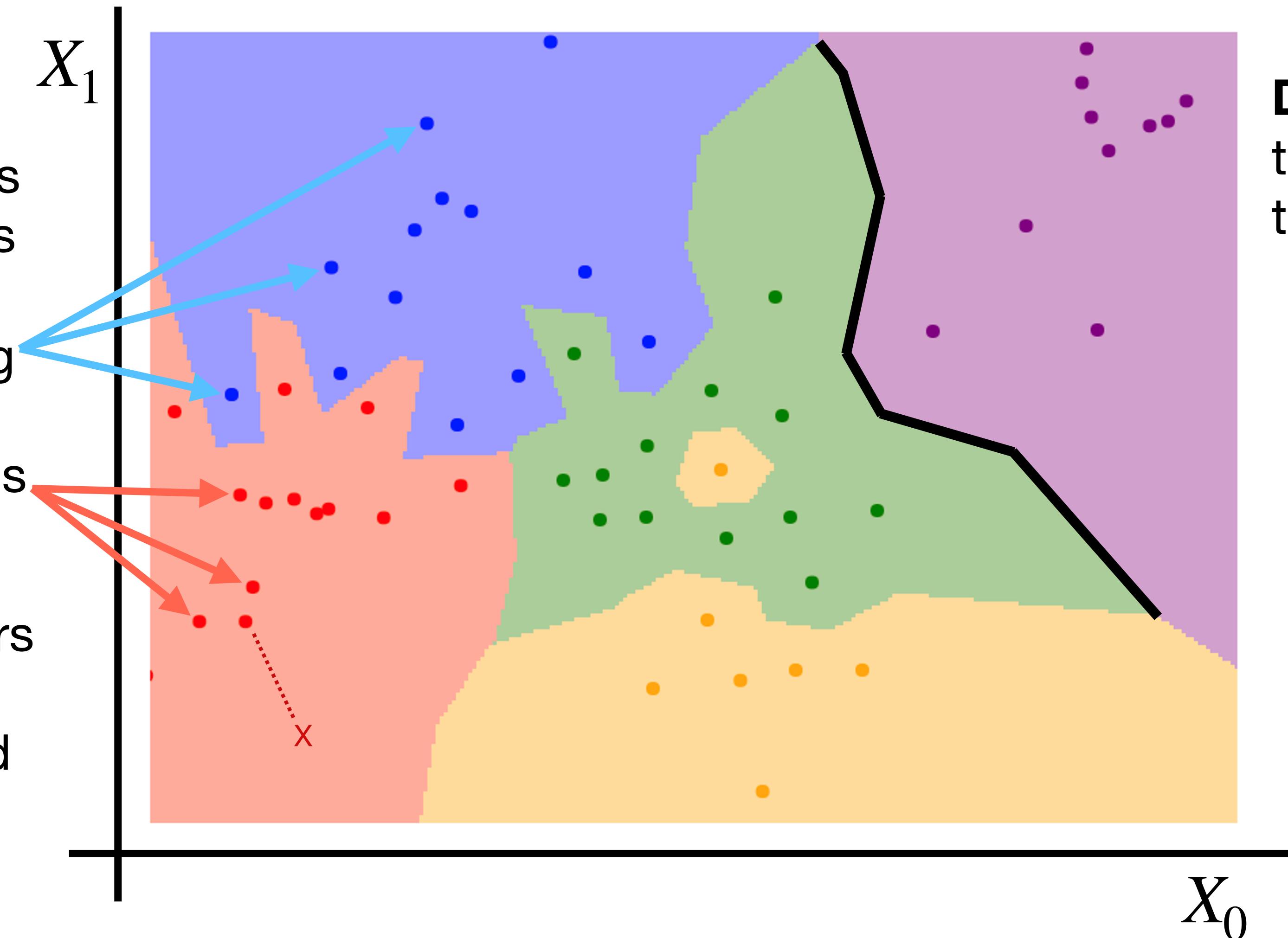


# K-Nearest Neighbors Decision Boundaries

Nearest neighbors  
in two dimensions

Points are training  
examples; colors  
give training labels

Background colors  
give the category  
a test point would  
be assigned



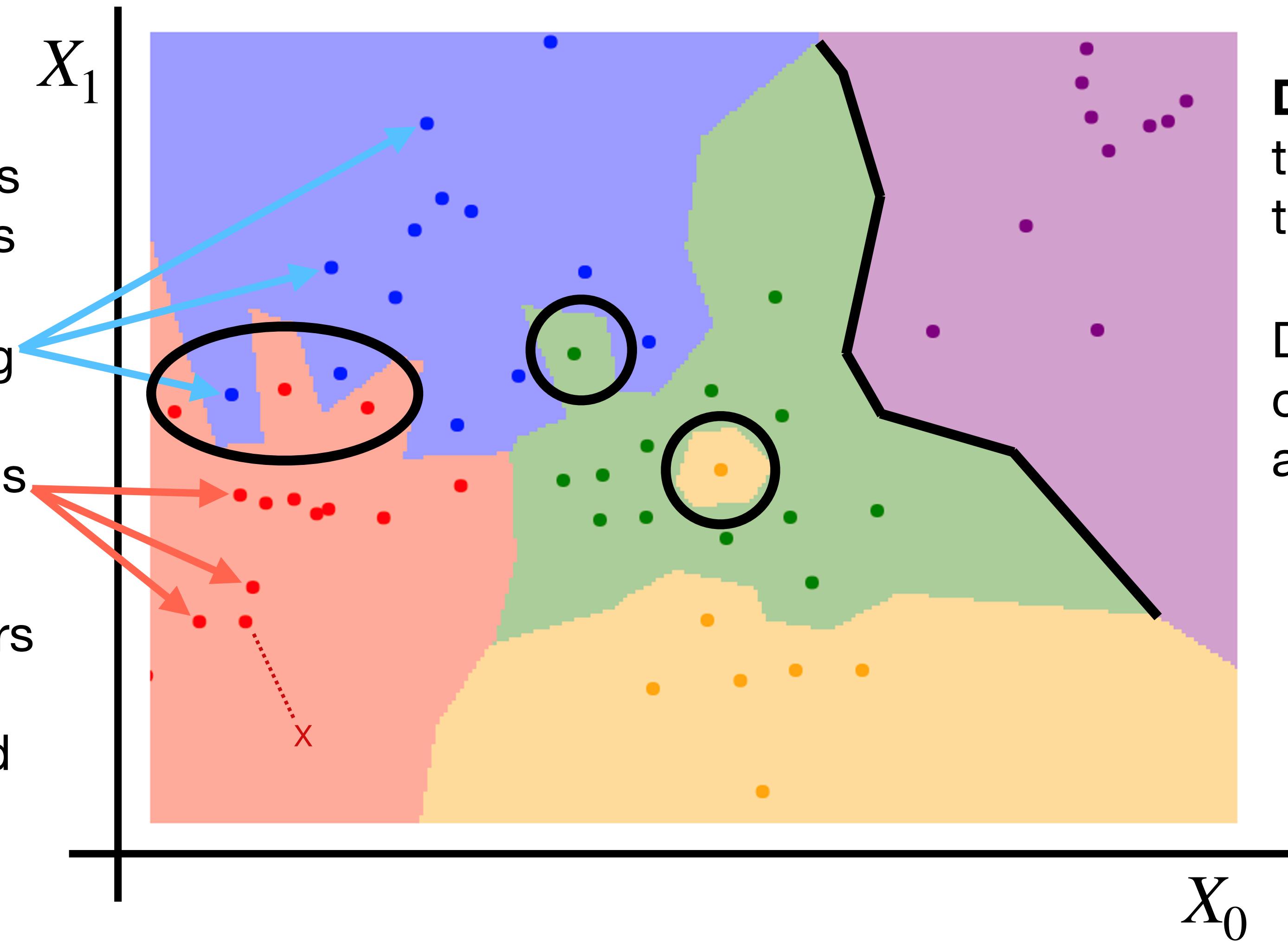
**Decision boundary** is  
the boundary between  
two classification regions

# K-Nearest Neighbors Decision Boundaries

Nearest neighbors  
in two dimensions

Points are training  
examples; colors  
give training labels

Background colors  
give the category  
a test point would  
be assigned



**Decision boundary** is  
the boundary between  
two classification regions

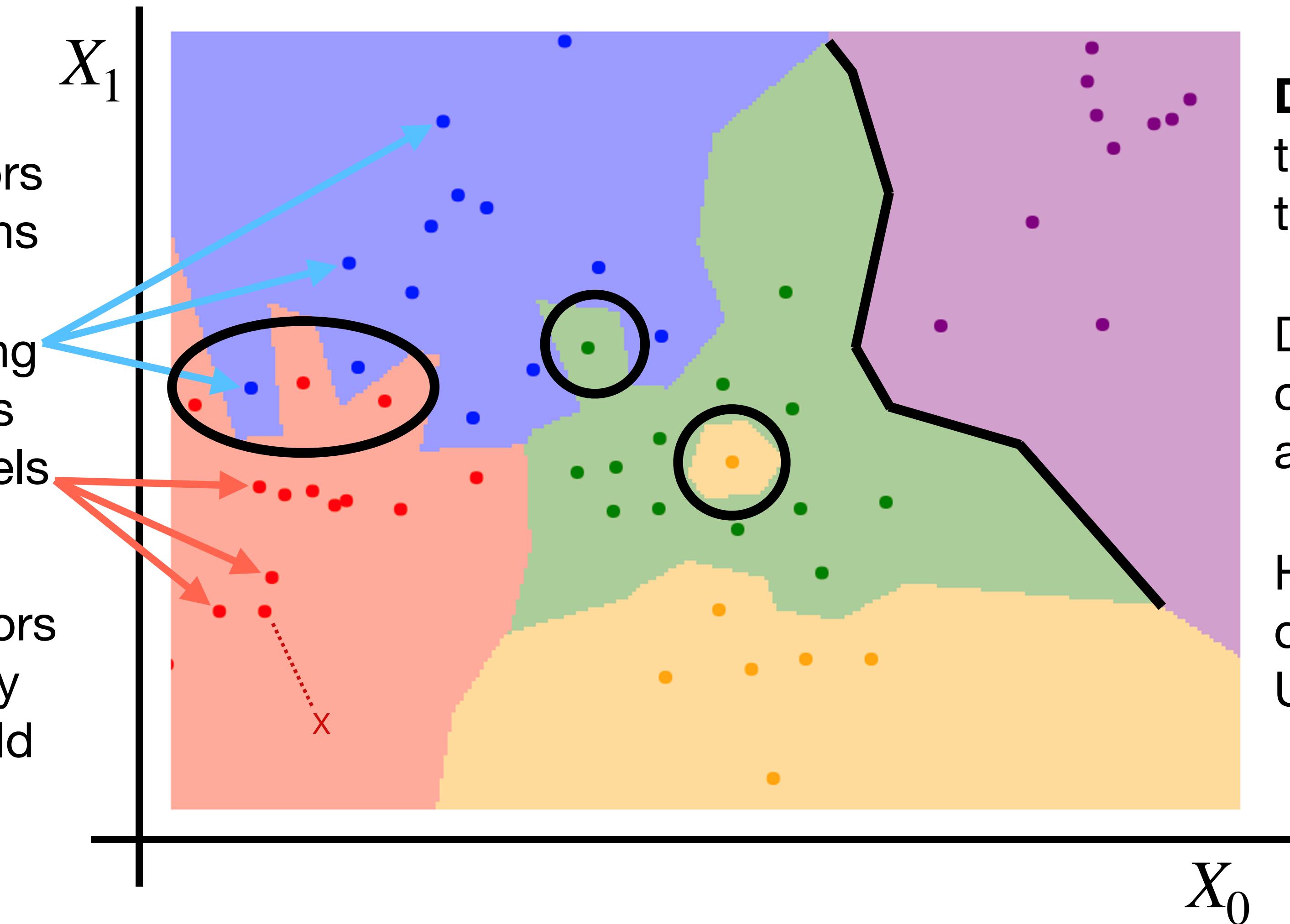
Decision boundaries  
can be noisy;  
affected by outliers

# K-Nearest Neighbors Decision Boundaries

Nearest neighbors  
in two dimensions

Points are training  
examples; colors  
give training labels

Background colors  
give the category  
a test point would  
be assigned



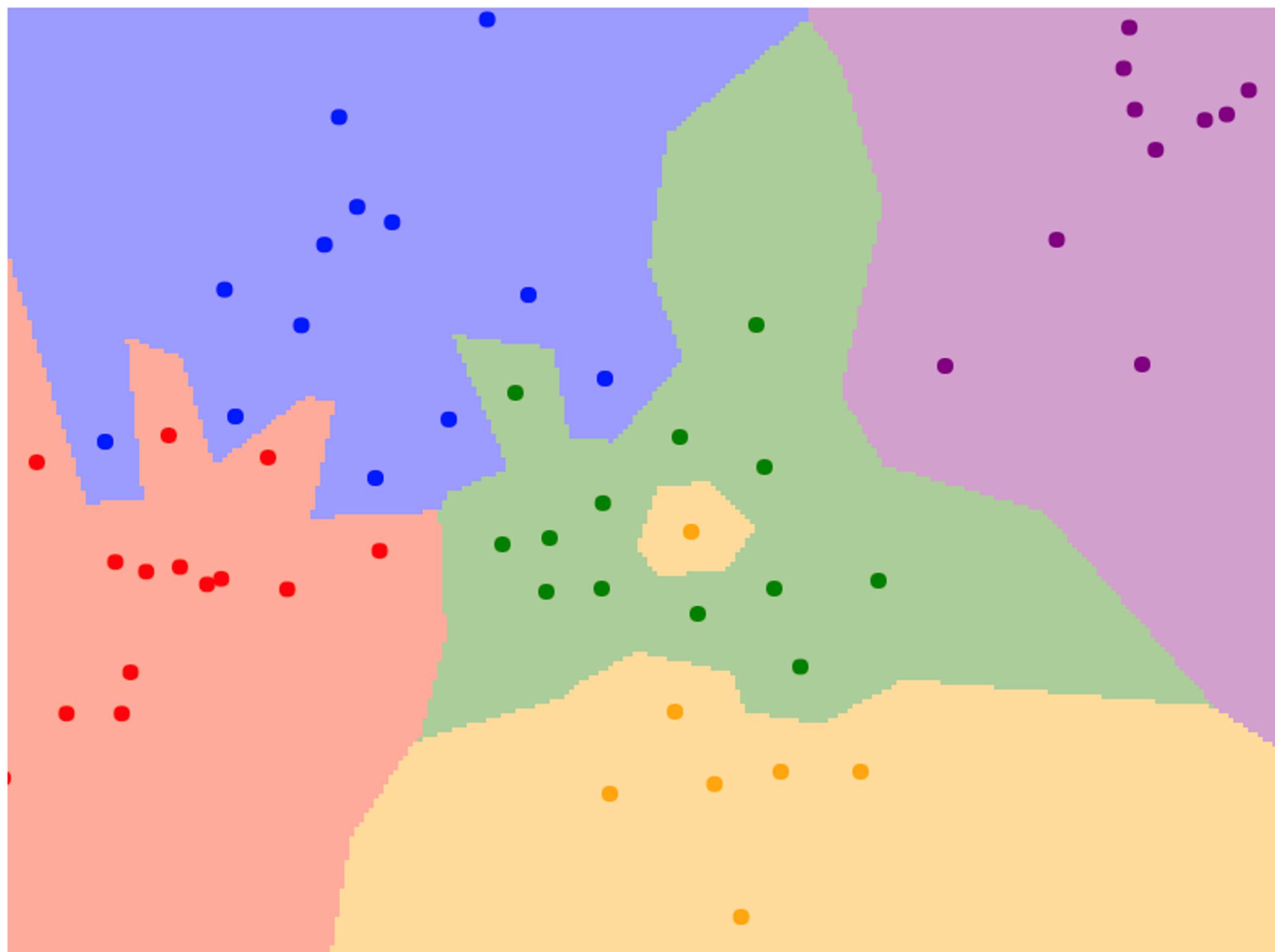
**Decision boundary** is  
the boundary between  
two classification regions

Decision boundaries  
can be noisy;  
affected by outliers

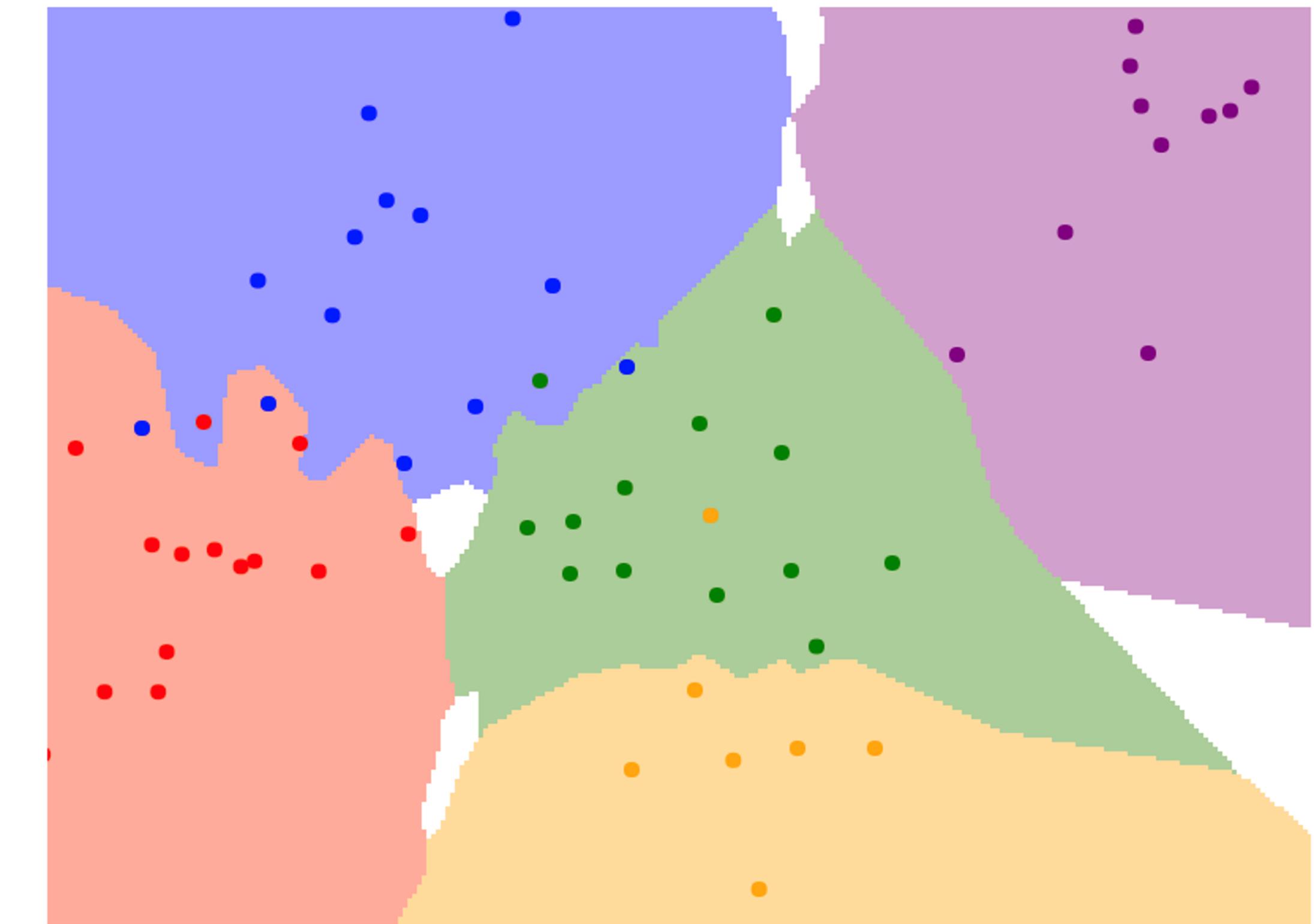
How to smooth the  
decision boundaries?  
Use more neighbors!

# K-Nearest Neighbors Classification

$K = 1$



$K = 3$

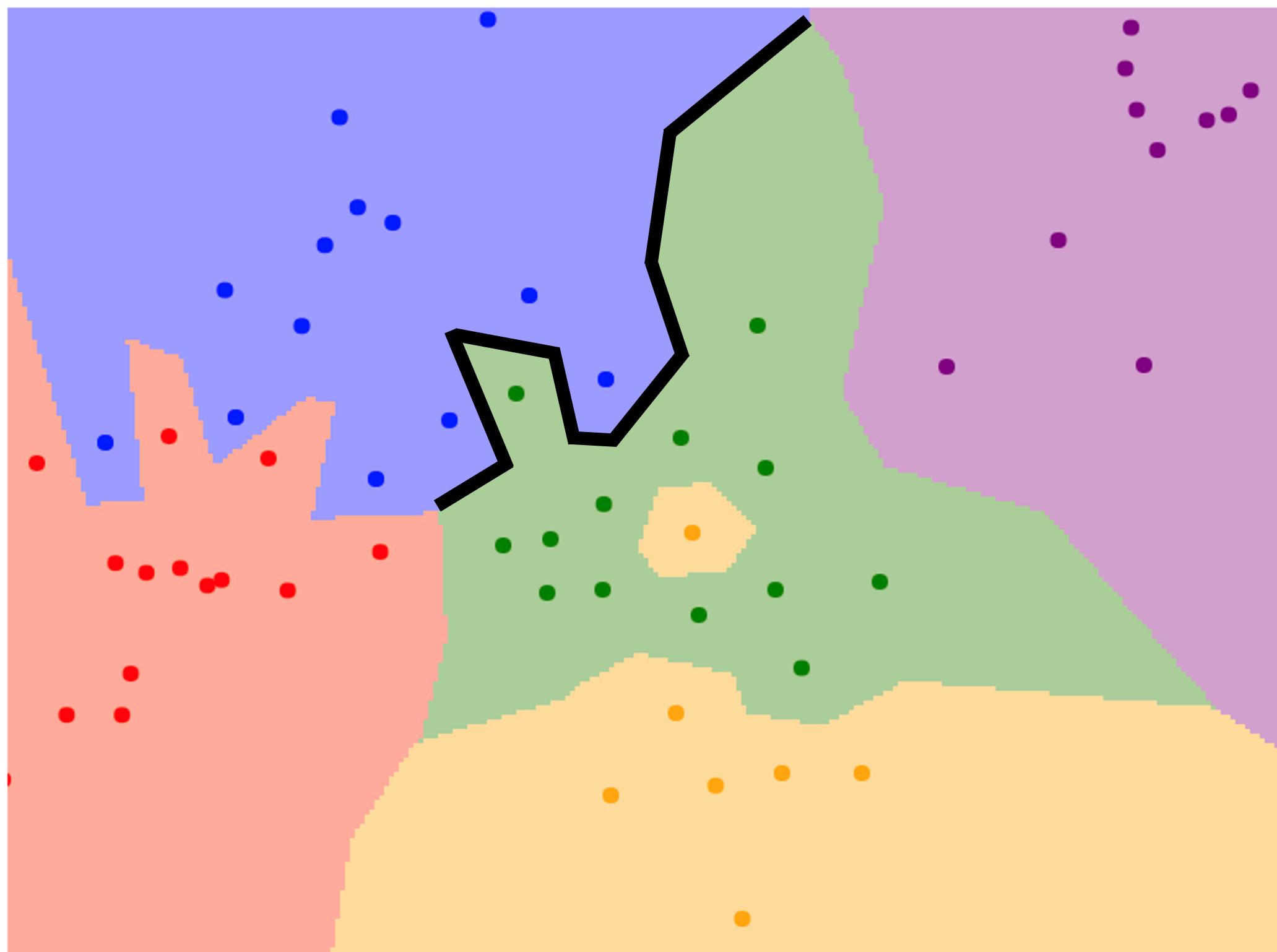


Instead of copying label from nearest neighbor,  
take majority vote from  $K$  closest training points

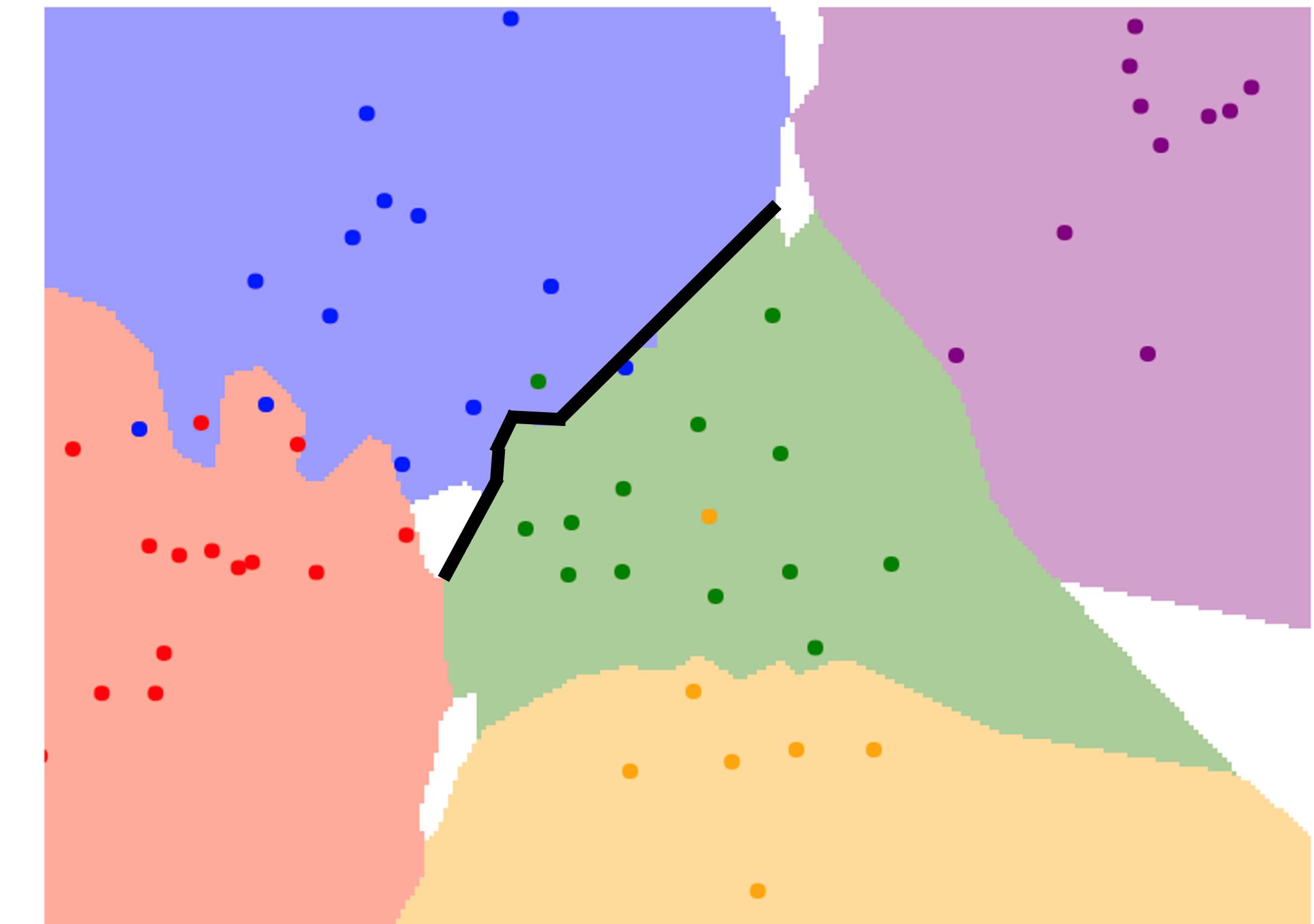


# K-Nearest Neighbors Classification

$K = 1$



$K = 3$

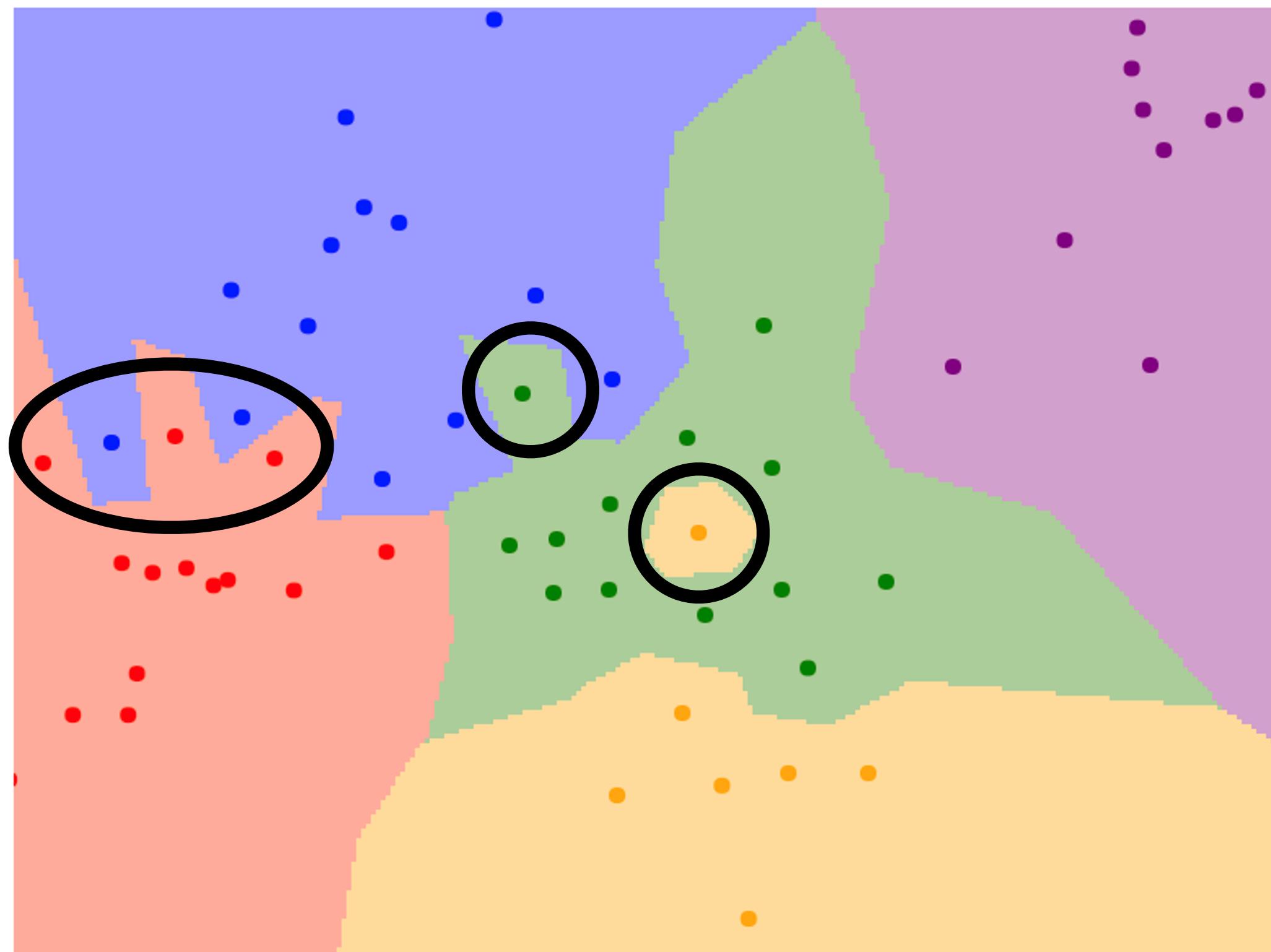


Using more neighbors helps smooth out rough decision boundaries

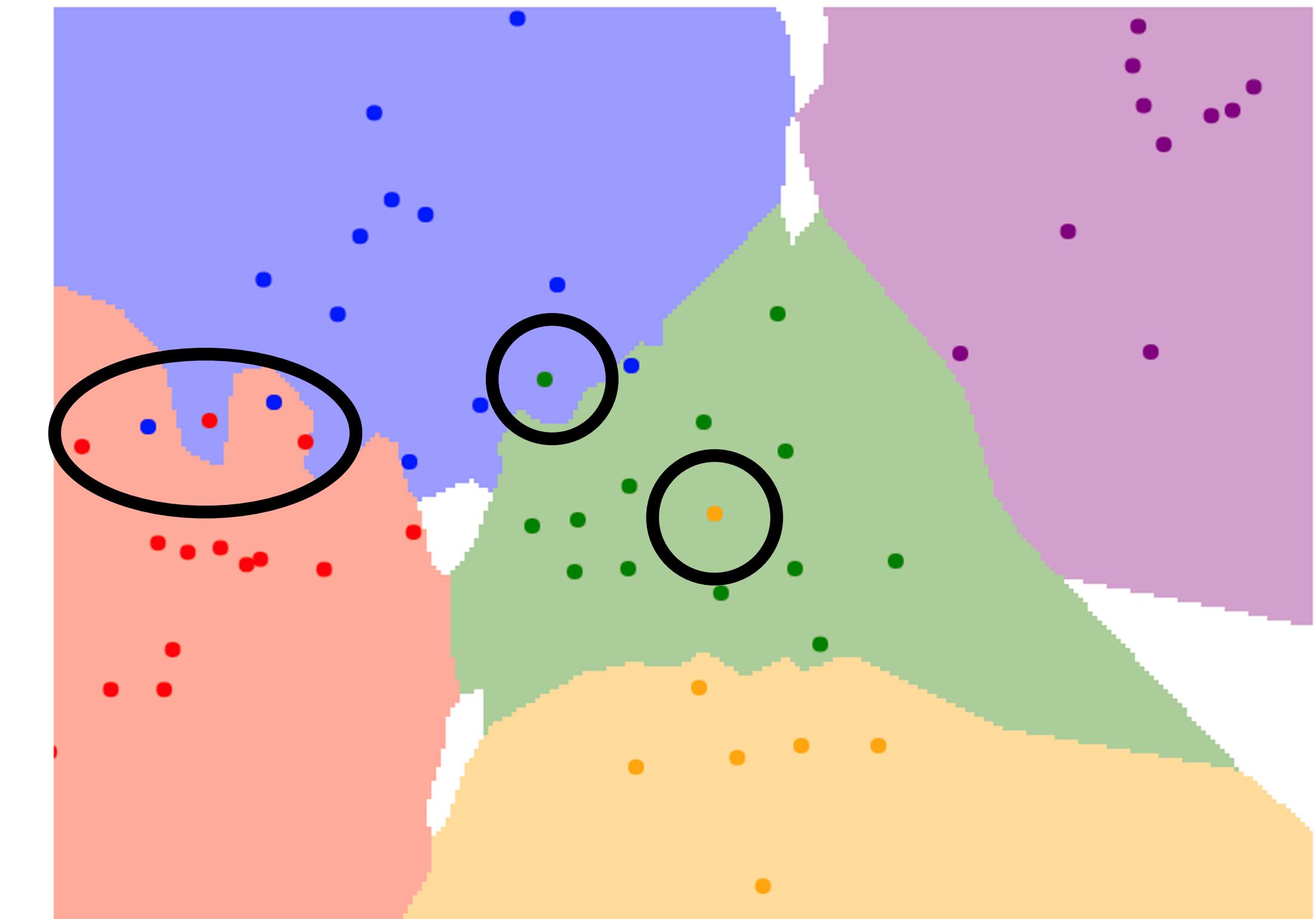


# K-Nearest Neighbors Classification

$K = 1$



$K = 3$

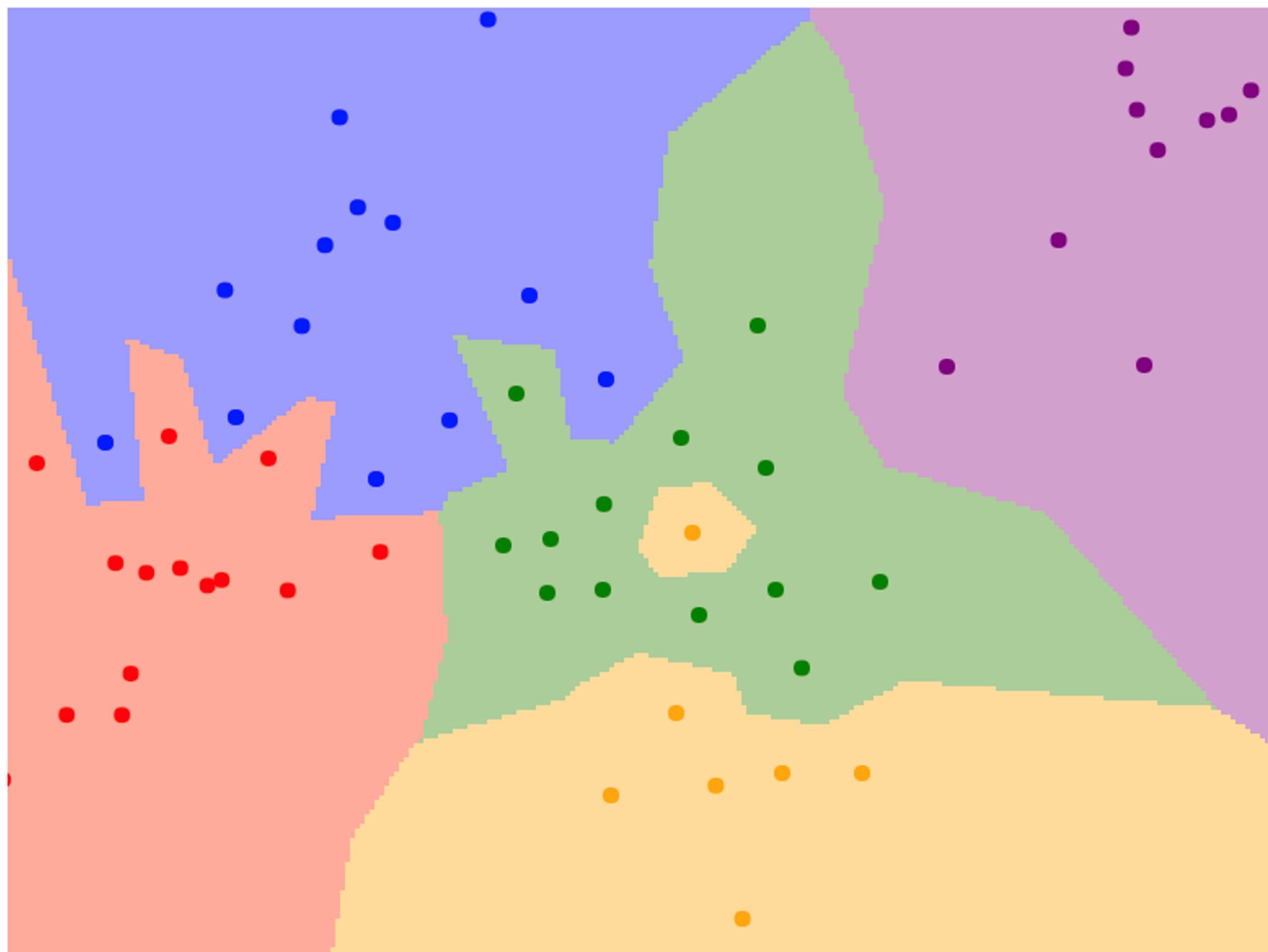


Using more neighbors helps reduce the effect of outliers

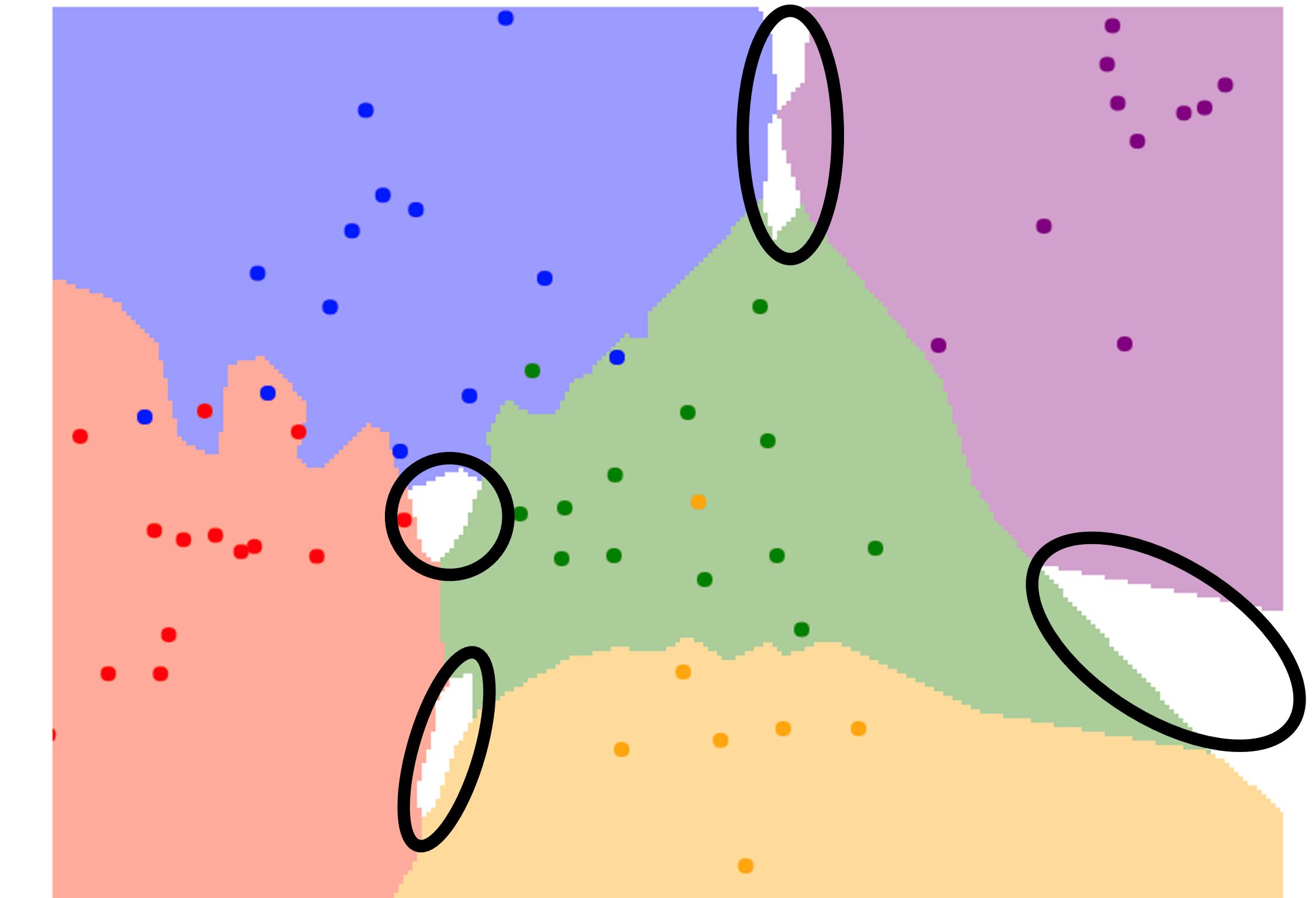


# K-Nearest Neighbors Classification

$K = 1$



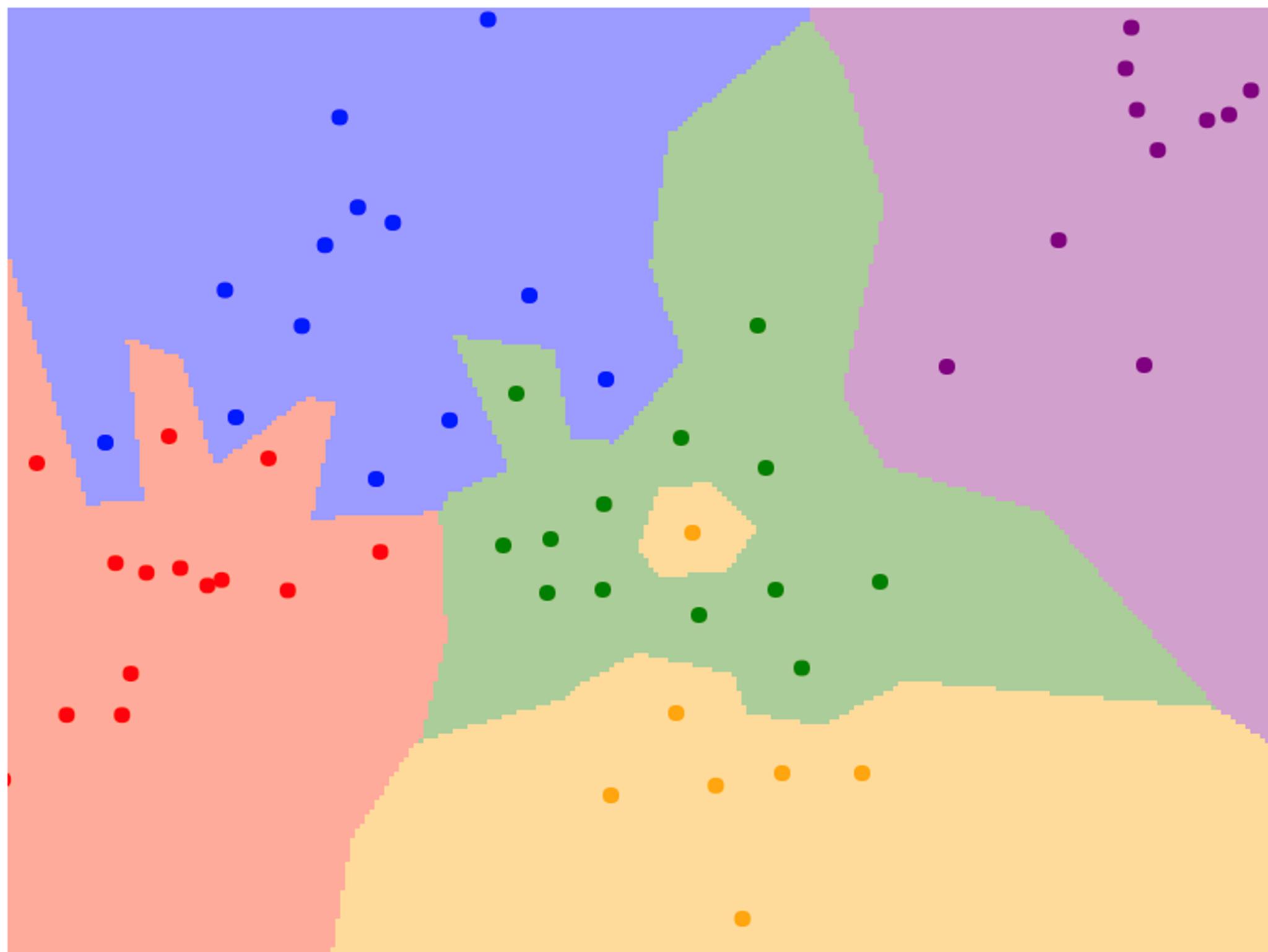
$K = 3$



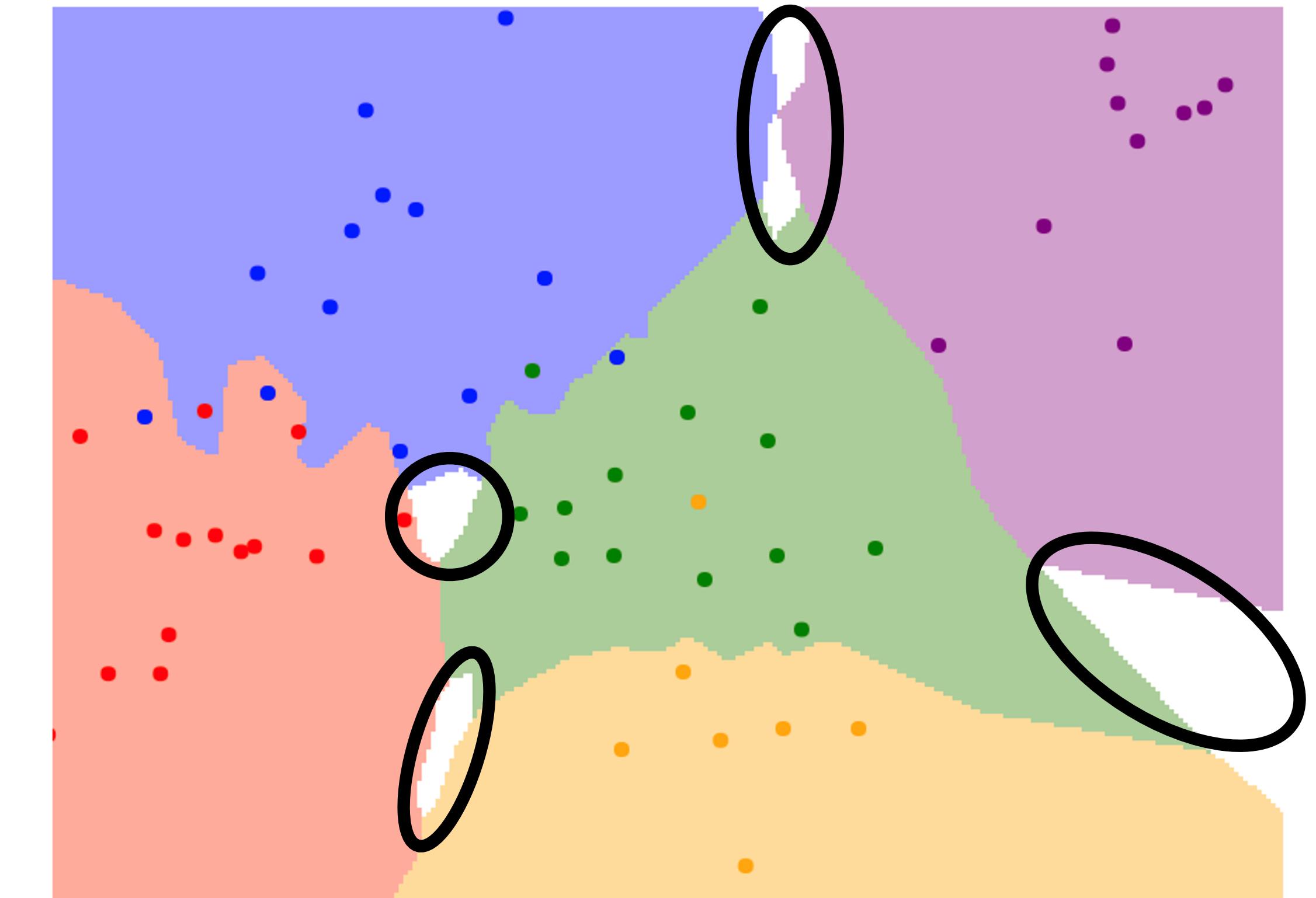
When  $K > 1$  there can be ties between classes.  
Need to break ties somehow!

# K-Nearest Neighbors Classification

$K = 1$



$K = 3$



When  $K > 1$  there can be ties between classes.  
Need to break ties somehow!

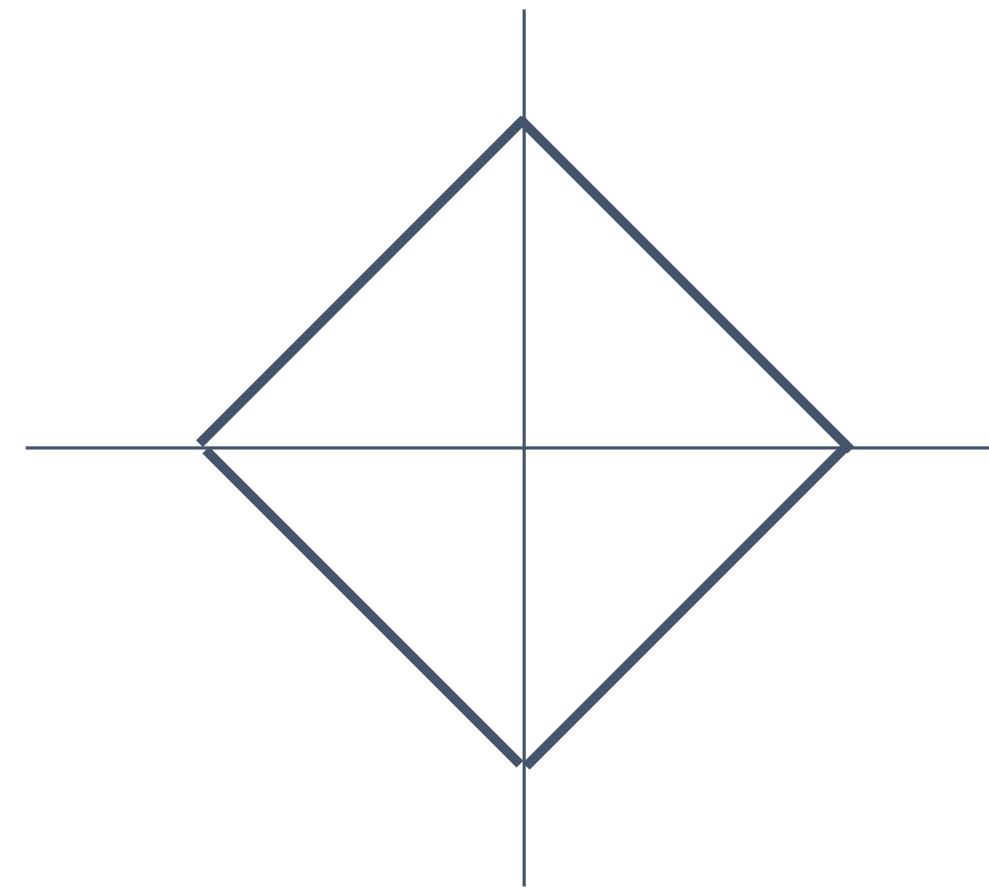


# K-Nearest Neighbors—Distance Metric

---

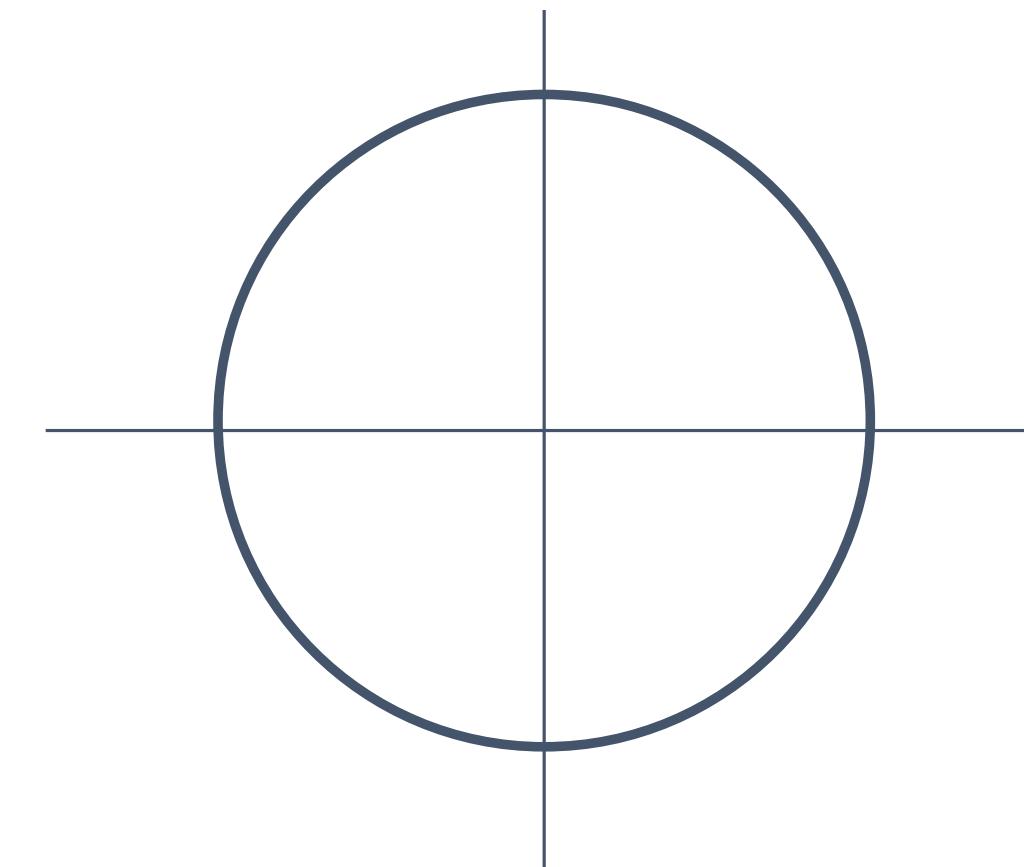
L1 (Manhattan) distance

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$



L2 (Euclidean) distance

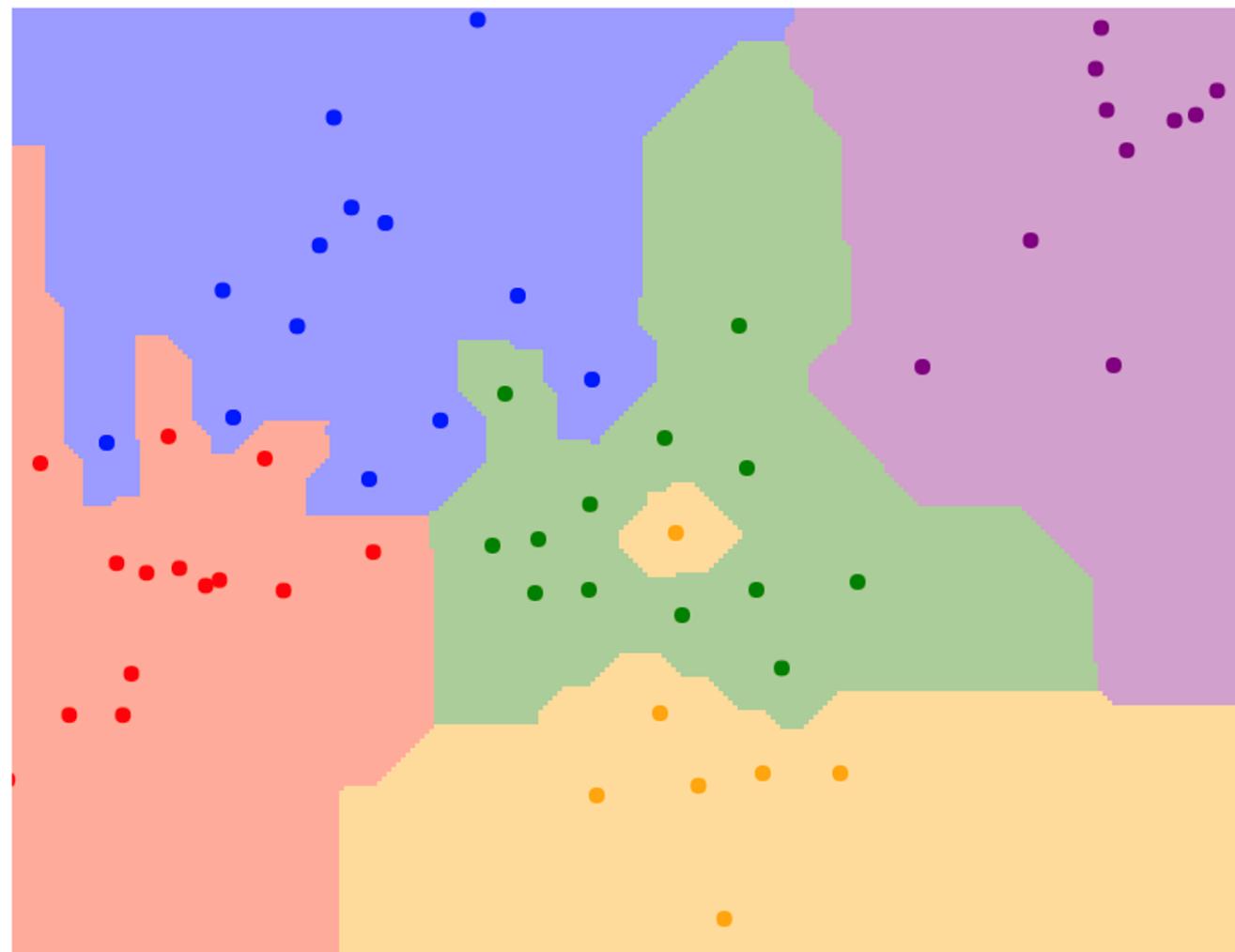
$$d_2(I_1, I_2) = (\sum_p (I_1^p - I_2^p)^2)^{\frac{1}{2}}$$



# K-Nearest Neighbors—Distance Metric

L1 (Manhattan) distance

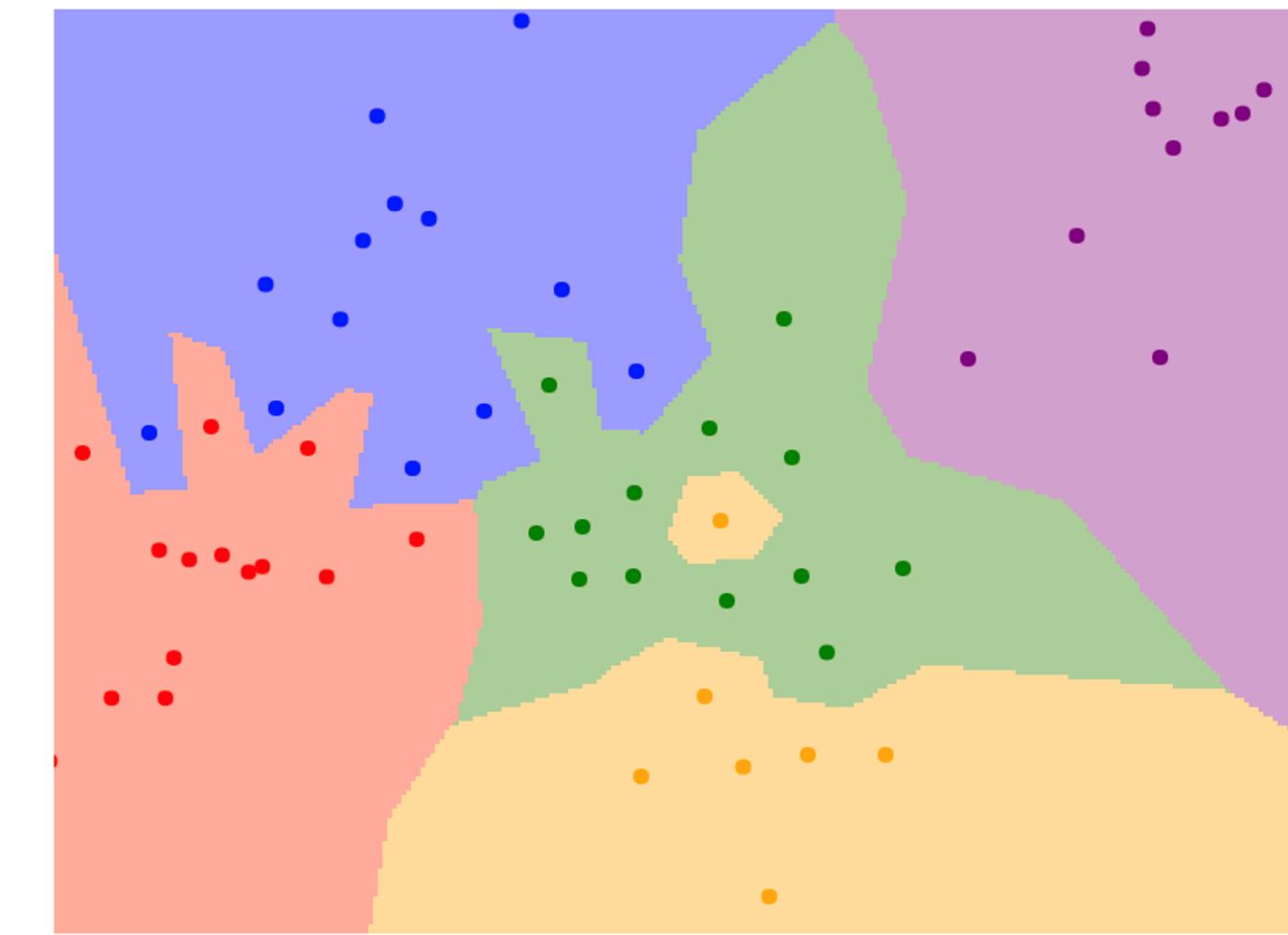
$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$



$$K = 1$$

L2 (Euclidean) distance

$$d_2(I_1, I_2) = (\sum_p (I_1^p - I_2^p)^2)^{\frac{1}{2}}$$



# K-Nearest Neighbors—Distance Metric

---

With the right choice of distance metric, we can apply K-Nearest Neighbors to any type of data!

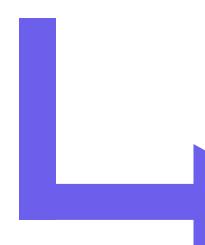


# K-Nearest Neighbors—Web Demo

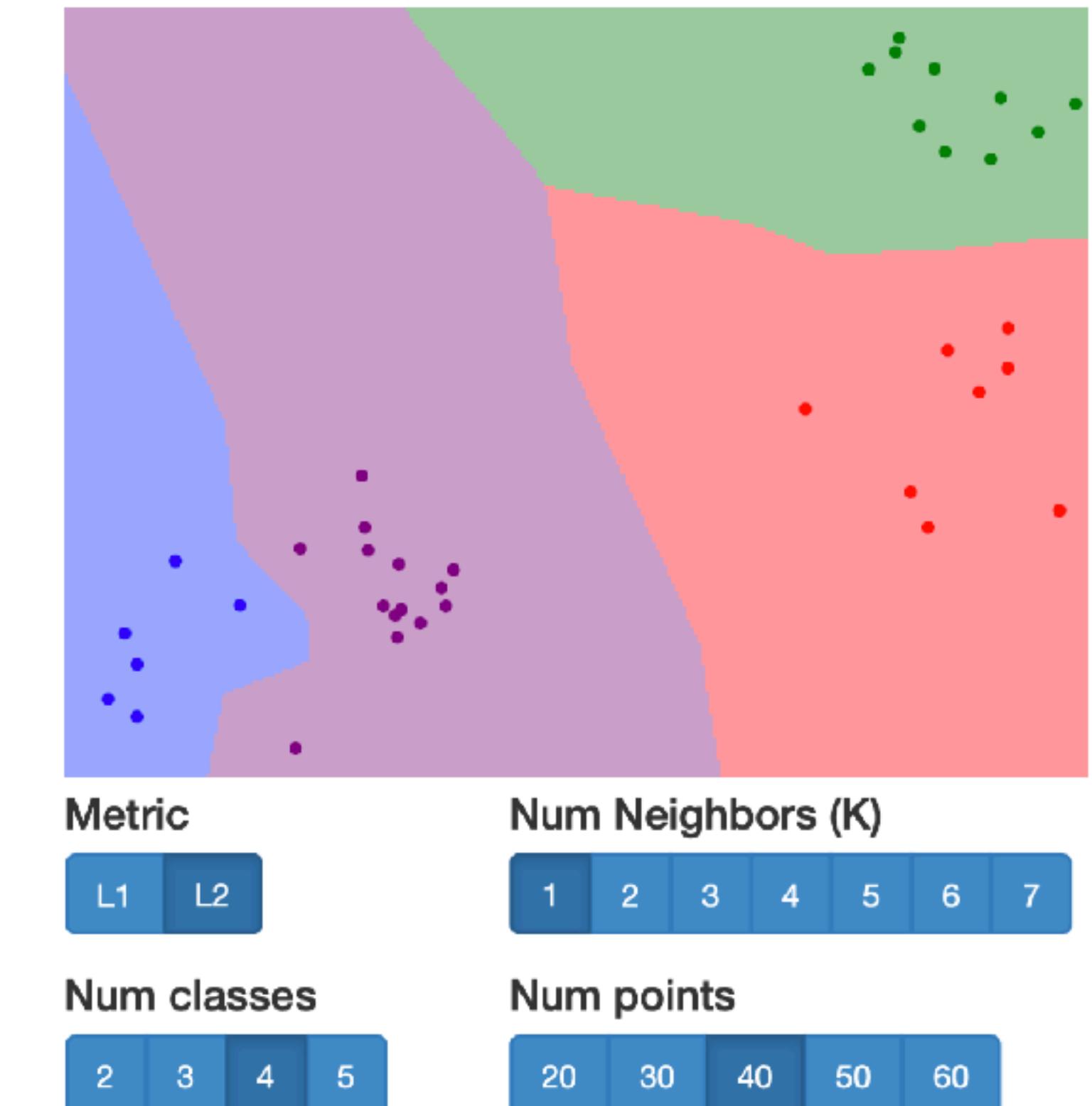
Interactively move points around  
and see decision boundaries change

Observe results with L1 vs L2 metrics

Observe results with changing number  
of training points and value of  $K$



<http://vision.stanford.edu/teaching/cs231n-demos/knn/>



# Hyperparameters

---

What is the best value of  $K$  to use?

What is the best **distance metric** to use?



# Hyperparameters

---

What is the best value of  $K$  to use?

What is the best **distance metric** to use?

These are examples of **hyperparameters**:

choices about our learning algorithm that we don't learn from the training data

Instead we set them at the start of the learning process



# Hyperparameters

---

What is the best value of  $K$  to use?

What is the best **distance metric** to use?

These are examples of **hyperparameters**:

choices about our learning algorithm that we don't learn from the training data

Instead we set them at the start of the learning process

Very problem-dependent.

In general need to try them all and observe what works best for our data.



# Setting Hyperparameters

---

**Idea #1:** Choose hyperparameters that work best on the data

Your Dataset



# Setting Hyperparameters

---

**Idea #1:** Choose hyperparameters that work best on the data

BAD:  $K = 1$  always works perfectly on training data

Your Dataset



# Setting Hyperparameters

**Idea #1:** Choose hyperparameters that work best on the data

BAD:  $K = 1$  always works perfectly on training data

Your Dataset

**Idea #2:** Split data into **train** and **test**, choose hyperparameters that work best on test data



# Setting Hyperparameters

**Idea #1:** Choose hyperparameters that work best on the data

**BAD:**  $K = 1$  always works perfectly on training data

Your Dataset

**Idea #2:** Split data into **train** and **test**, choose hyperparameters that work best on test data

**BAD:** No idea how algorithm will perform on new data

train

test



# Setting Hyperparameters

**Idea #1:** Choose hyperparameters that work best on the data

BAD:  $K = 1$  always works perfectly on training data

Your Dataset

**Idea #2:** Split data into **train** and **test**, choose hyperparameters that work best on test data

BAD: No idea how algorithm will perform on new data

train

test

**Idea #3:** Split data into **train**, **val**, and **test**; choose hyperparameters on val and evaluate on test

**Better!**

train

validation

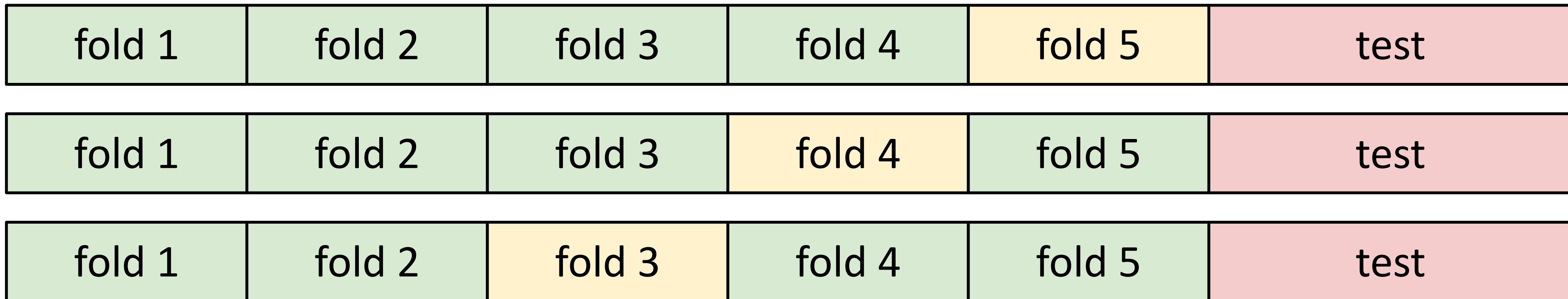
test



# Setting Hyperparameters

Your Dataset

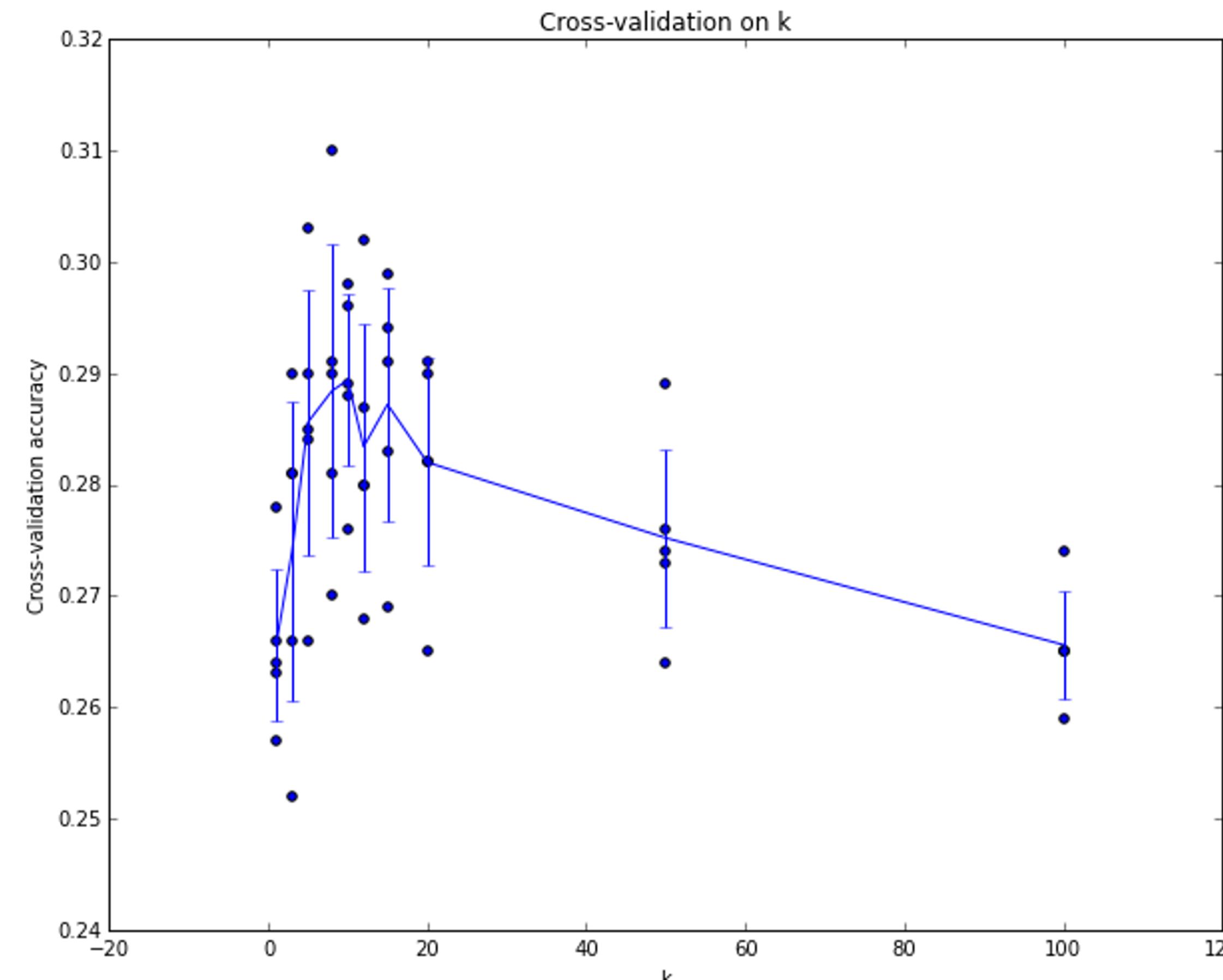
**Idea #4: Cross-Validation:** Split data into **folds**, try each fold as validation and average the results



Useful for small datasets, but (unfortunately) not used too frequently in deep learning



# Setting Hyperparameters



Example of 5-fold cross-validation for the value of k.

Each point: single outcome.

The line goes through the mean, bars indicated standard deviation

(Seems that  $k \sim 7$  works best for this data)

# K-Nearest Neighbors—Universal Approximation

---

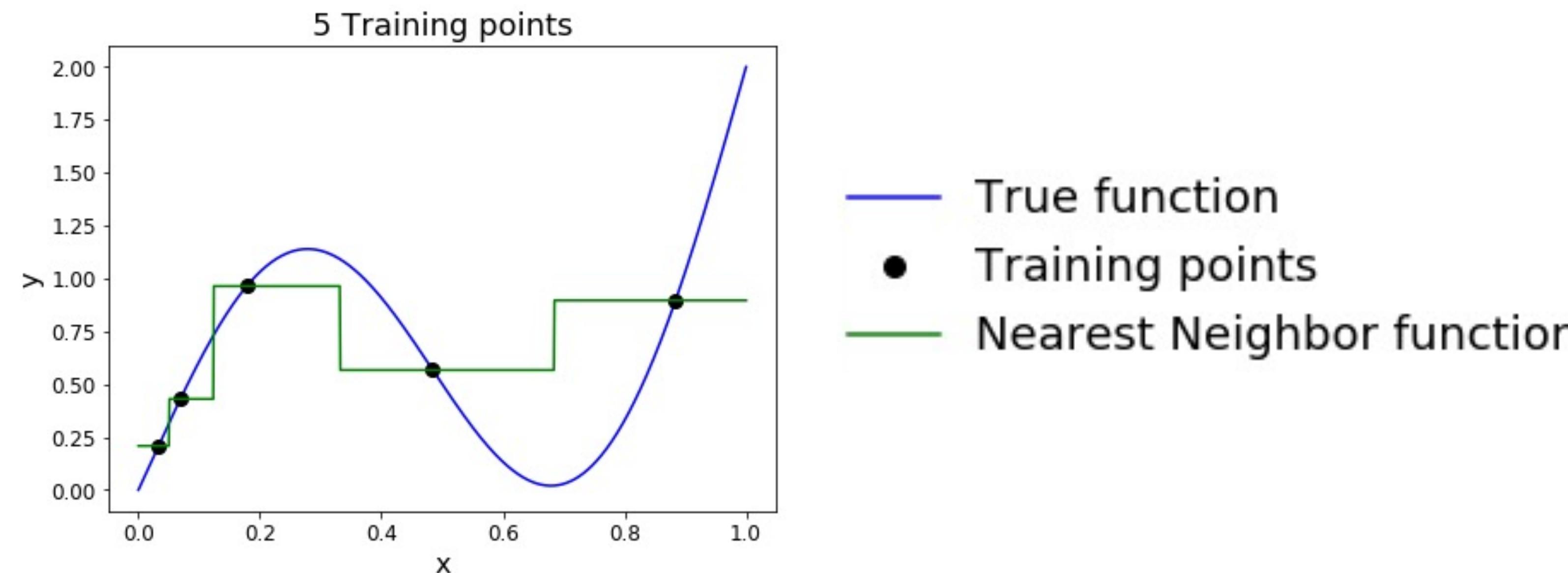
As the number of training samples goes to infinity, nearest neighbor can represent any<sup>(\*)</sup> function!



(\*) Subject to many technical conditions. Only continuous functions on a compact domain; need to make assumptions about spacing of training points; etc.

# K-Nearest Neighbors—Universal Approximation

As the number of training samples goes to infinity, nearest neighbor can represent any<sup>(\*)</sup> function!

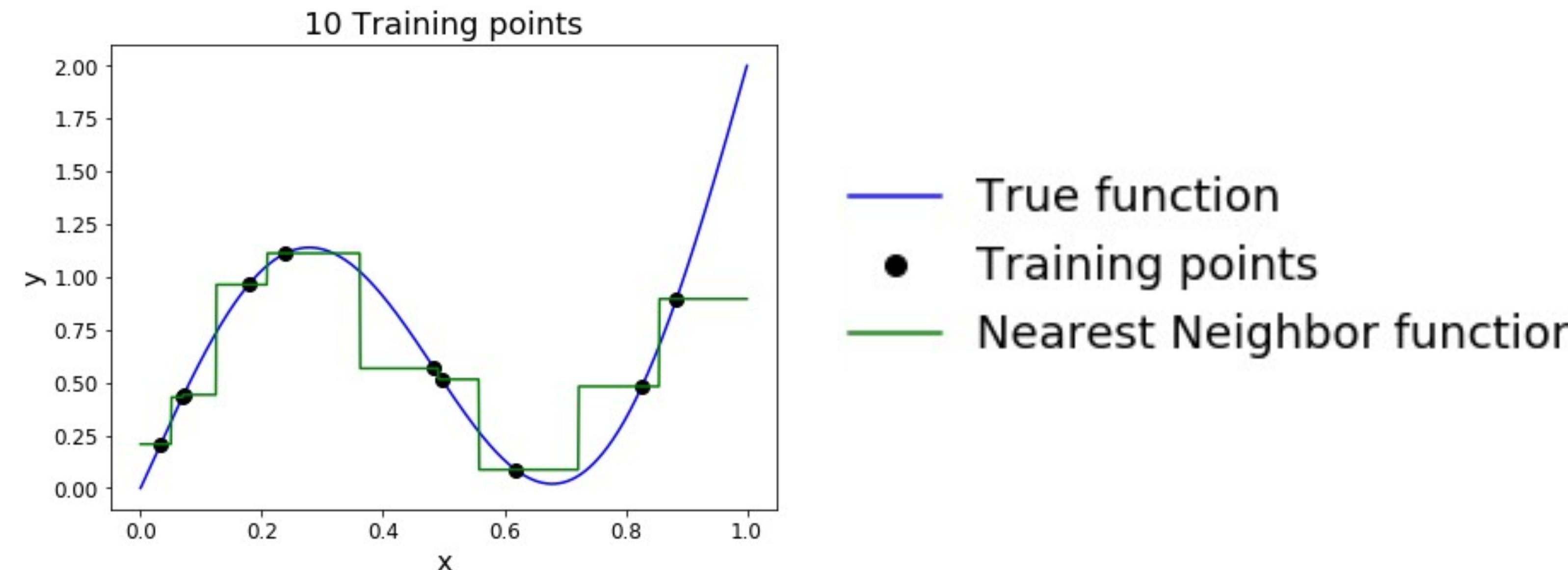


(\*) Subject to many technical conditions. Only continuous functions on a compact domain; need to make assumptions about spacing of training points; etc.



# K-Nearest Neighbors—Universal Approximation

As the number of training samples goes to infinity, nearest neighbor can represent any<sup>(\*)</sup> function!

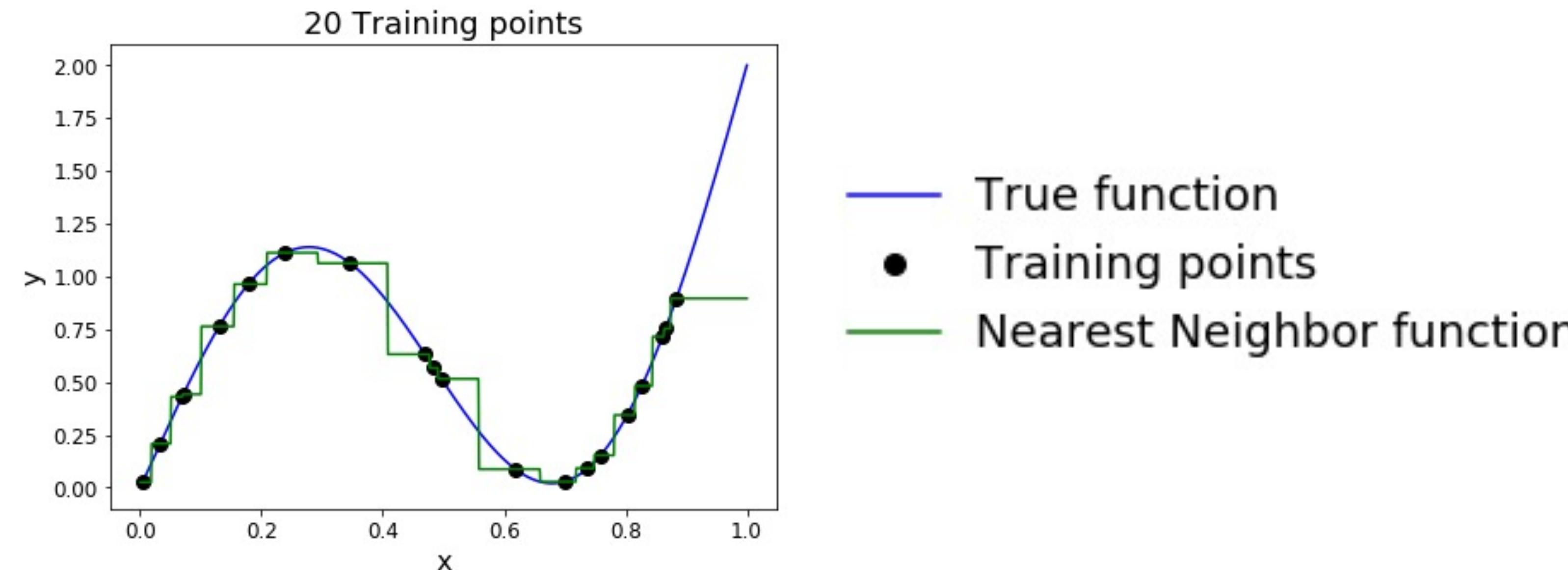


(\*) Subject to many technical conditions. Only continuous functions on a compact domain; need to make assumptions about spacing of training points; etc.



# K-Nearest Neighbors—Universal Approximation

As the number of training samples goes to infinity, nearest neighbor can represent any<sup>(\*)</sup> function!

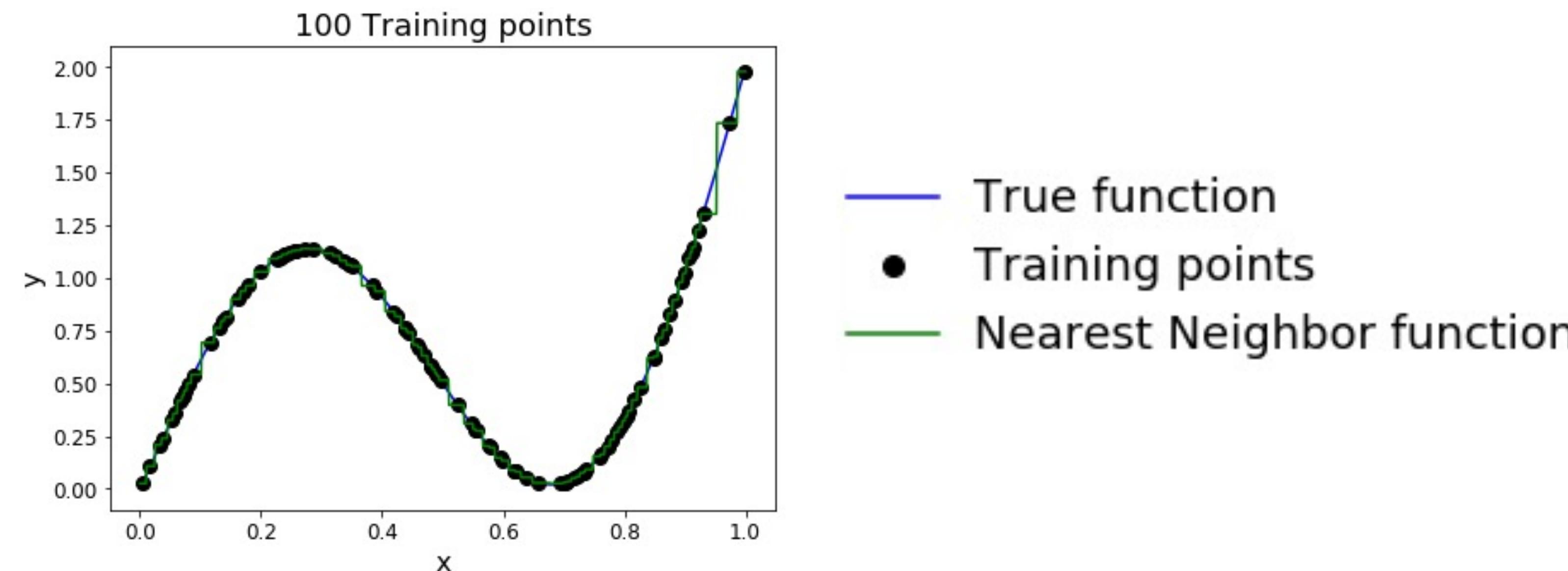


(\*) Subject to many technical conditions. Only continuous functions on a compact domain; need to make assumptions about spacing of training points; etc.



# K-Nearest Neighbors—Universal Approximation

As the number of training samples goes to infinity, nearest neighbor can represent any<sup>(\*)</sup> function!

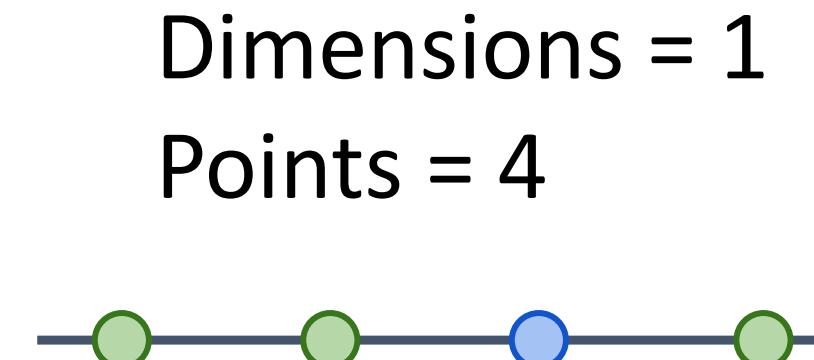


(\*) Subject to many technical conditions. Only continuous functions on a compact domain; need to make assumptions about spacing of training points; etc.

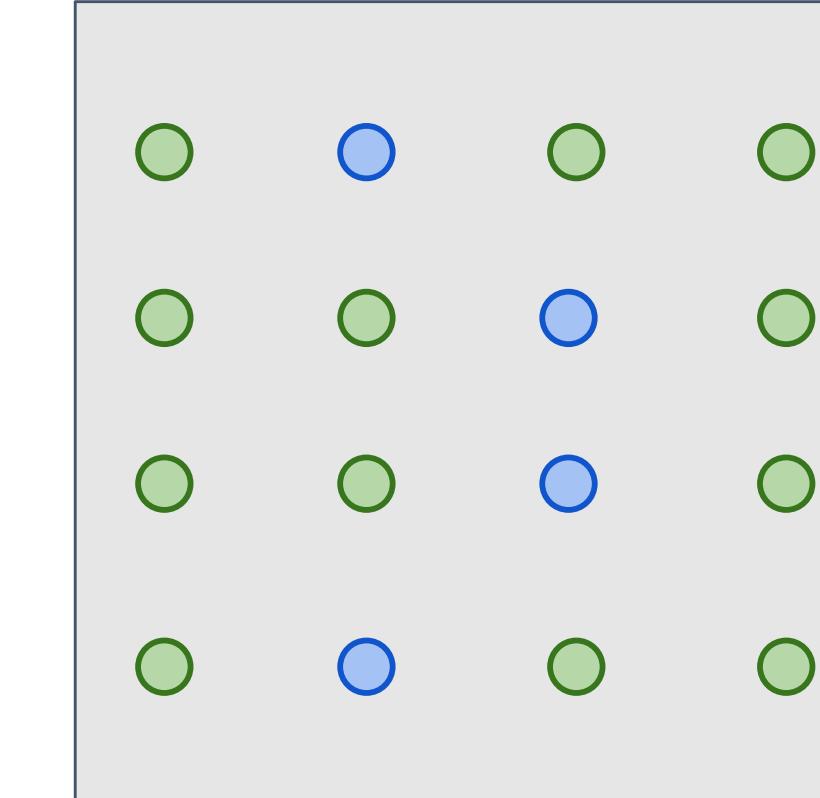


# Problem—Curse of Dimensionality

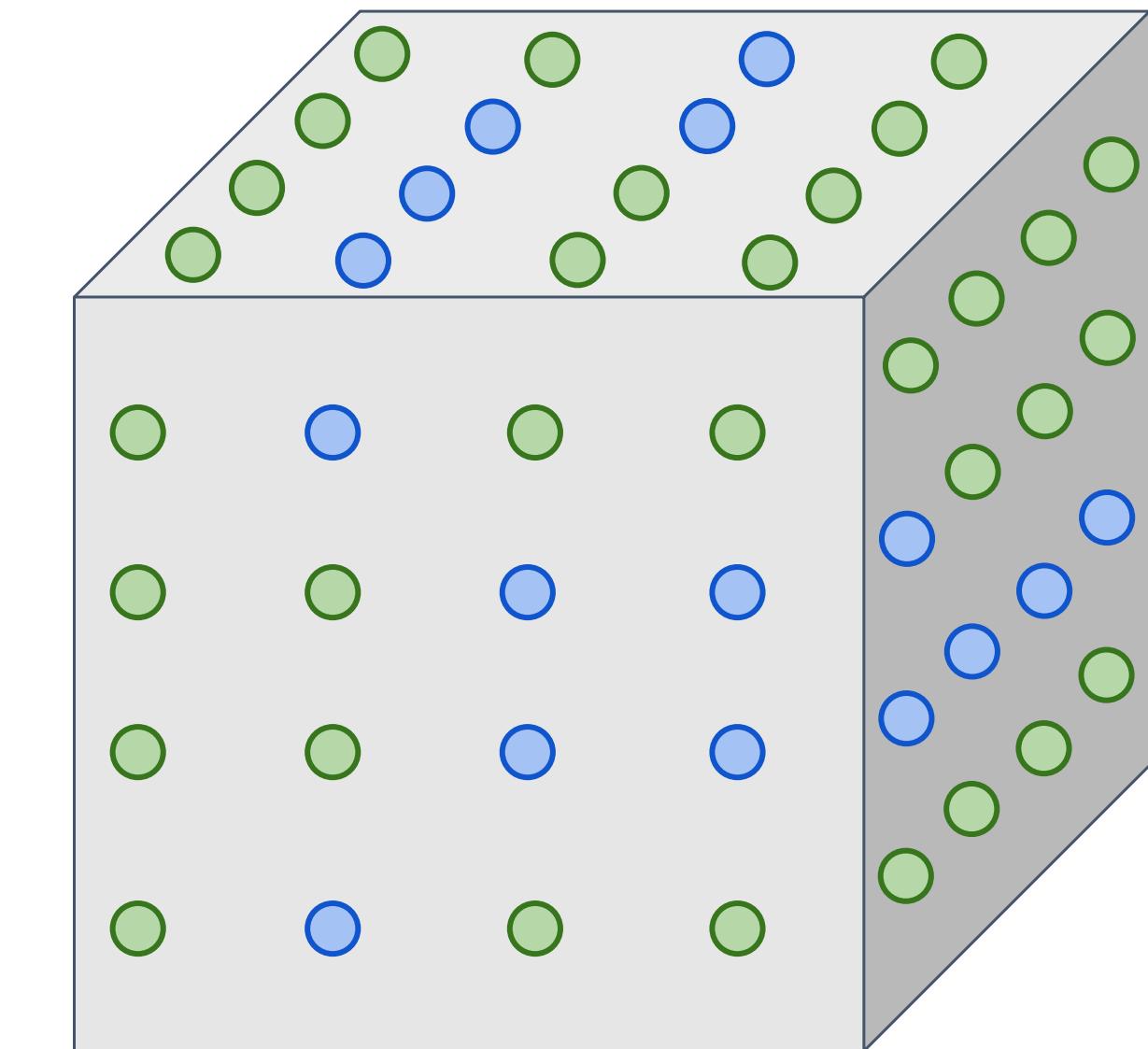
**Curse of dimensionality:** For uniform coverage of space, number of training points needed grows exponentially with dimension



Dimensions = 2  
Points =  $4^2$



Dimensions = 3  
Points =  $4^3$



# Problem—Curse of Dimensionality

---

**Curse of dimensionality:** For uniform coverage of space, number of training points needed grows exponentially with dimension

Number of possible  
32x32 binary images

$$2^{32 \times 32} \approx 10^{308}$$



# Problem—Curse of Dimensionality

---

**Curse of dimensionality:** For uniform coverage of space, number of training points needed grows exponentially with dimension

Number of possible  
32x32 binary images

$$2^{32 \times 32} \approx 10^{308}$$

Number of elementary  
particles in the visible [source](#)

$$\approx 10^{97}$$



# K-Nearest Neighbors Seldom Used on Raw Pixels

Very slow at test time

Distance metrics on pixels are not informative



All 3 images have same L2 distance to the original

# K-Nearest Neighbors with Convent Features Works Well



Devlin et al., “Exploring Nearest Neighbor Approaches for Image Captioning”, 2015.

# Summary

---

In **image classification** we start with a training set of images and labels, and must predict labels for a test set

Image classification is challenging due to the **semantic gap**: we need invariance to occlusion, deformation, lighting, sensor variation, etc.

Image classification is a **building block** for other vision tasks

The **K-Nearest Neighbors** classifier predicts labels from nearest training samples

Distance metric and  $K$  are **hyperparameters**

Choose hyper parameters using the **validation set**; only run on the test set once at the very end!



# Next time: Linear Classifiers

